Broadview
www.broadview.com.cn

走进搜索引擎

Stepping into Search Engine

梁 斌 编著

打造优质搜索引擎的第一书!

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

VISIT…

LANZAROTE Caliente.com

走进搜索引擎

Stepping into Search Engine

梁斌 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

在网络普及的今天，人们经常在信息海洋中彷徨，在万维网迷宫般的复杂与魅力之间挣扎。直到搜索引擎这一伟大的技术产生，才使得人们犹如找到了走出迷宫的灯塔，可以非常便捷地找到自己所需要的信息。

正是因为搜索引擎离我们越来越近，所以越来越多的人期待着能够揭开她神秘的面纱。其实搜索引擎并不是变幻莫测的大海，也不是高不可攀的山峰。请拿起本书，它就是引领你的火炬，它就是你身边的伙伴，它将带着你走进搜索引擎。在那里，你必将会被搜索引擎精致的设计和宏伟的架构所征服。

本书由搜索引擎开发研究领域年轻而有活力的科学家精心编写，作者将自己对搜索引擎的深刻理解和实际应用巧妙地结合，使得从未接触过搜索引擎原理的读者也能够轻松地在搜索引擎的大厦中遨游一番。

本书作为搜索引擎原理与技术的入门书籍，面向那些有志从事搜索引擎行业的青年学生、需要完整理解并优化搜索引擎的专业技术人员、搜索引擎的营销人员，以及网站的负责人等。

本书是从事搜索引擎开发的工程技术人员难得的参考书，也可作为大中专院校相关专业的教学辅导书。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

走进搜索引擎 / 梁斌编著.—北京：电子工业出版社，2007.10
ISBN 978-7-121-04922-4

I. 走… II.梁… III.互联网络－情报检索 IV.G354.4

中国版本图书馆 CIP 数据核字（2007）第 132658 号

责任编辑： 孙学瑛
印　　刷： 北京智力达印刷有限公司
装　　订： 北京中新伟业印刷有限公司
出版发行： 电子工业出版社出版
北京市海淀区万寿路 173 信箱　邮编：100036
开　　本： 787×980　1/16　印张：18.25　字数：258 千字
印　　次： 2007 年 10 月第 1 次印刷
印　　数： 5000 册　　定价：49.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

推荐序

搜索改变生活

搜索引擎，改变了二十一世纪人类的生活方式。越来越多的人学会通过搜索，从海量的互联网信息中找到和分享全人类的经验与智慧。搜索引擎不仅成为人们最常使用的互联网应用，同时也开创了一种优秀的商业模式，引领着互联网技术与商业的发展。

搜索引擎有着对计算机科学与技术孜孜不倦的追求，将人工智能、信息检索、文本处理、系统结构、数据库技术等都发挥到了极致，以追求响应更快、结果更准确。如同制造原子弹或者登月计划一样，搜索引擎在科技方面的突破也会辐射到互联网的其他应用领域。

在 Google、Yahoo 等国外公司投入巨资进入搜索引擎应用领域之时，中国本土公司百度、搜狗、有道，也不断加大投入，努力为中国用户打造更优秀的中文搜索引擎。能够有能力、有信念从事这样研发的中国公司不是太多，而是太少了；而对技术有着狂热爱好，能够将搜索引擎研发作为事业来看待的人才也不是太多，而是太少了。

我经常在技术论坛上看到这样的提问："现在的搜索引擎应该怎样设计数据库的表结构呢？（言下之意，还在考虑通用的关系数据库）""搜索引擎是在用户搜索的时候，怎么能够这么快抓取其他网站的信息呢？（看起来还没有理解索引和镜像的概念）"。这些提问充分显示出大多数技术人员对搜索引擎工作原理的不理解。

市面上，对搜索引擎原理进行讲解的书非常稀少。搜索引擎公司更将各自研发中的关键技术和经验作为最高机密不予公开。**我希望借助这本书，不仅可以让大众技术人员通过了解搜索引擎的工作原理，提高对技术的理解，更为那些想以搜索引擎研发为未来事业的人，打开一扇窗户。**

搜狐 技术副总裁

2007-9-1

前　　言

随着互联网的蓬勃发展，建立在互联网之上的各种应用也层出不穷，其中最为成功的莫过于万维网（WWW）。万维网被称为“网中之网”，是互联网上最受欢迎的服务之一。它运用超文本技术为人们访问信息资源提供了巨大的方便，但也以非线性组织的构建方式使人们在信息海洋中彷徨。

在互联网彷徨无计之时，搜索引擎就像引领互联网走出迷雾的灯塔。在搜索引擎的帮助下，网民不再需要记住复杂的网址和复杂的路径（URL），而只需记住搜索引擎的入口，提交查询词即可直接找到想要找的信息。就这样，搜索引擎成为破解互联网迷宫的金钥匙。

随着搜索引擎技术和商业模式的成熟，一方面，越来越多的人对搜索引擎产生了浓厚的兴趣，需要深入了解和认识搜索引擎；另一方面，由于搜索引擎是最高端、最复杂的互联网技术之一，各家公司都将核心技术秘而不宣。在这样的背景下，本书犹如一本引领读者走进搜索引擎的导游图，可以带领读者进入搜索引擎的腹地，一探搜索引擎神秘面纱的背后。那里是搜索引擎精致的设计和宏伟的架构，必定让您不虚此行。

本书主要内容

本书共包括 7 章，每章的主要内容如下。

第 1 章“引言”介绍了搜索引擎的基本背景知识，通过介绍搜索引擎的历史回顾了搜索引擎的发展历程。

第 2 章“搜索引擎概貌”宏观上介绍了搜索引擎，以及搜索引擎的主要系统划分。

第 3 章“搜索引擎的下载系统”介绍了搜索引擎下载系统的背景知识、设计原理和技巧，以及网页库的设计等。

第 4 章“搜索引擎的分析系统”分别介绍了信息抽取、网页查重、中文分词，以及 PageRank 等分析系统子模块的计算原理和实现细节。

第 5 章“搜索引擎的索引系统”通过全文检索、文档编号、正排表和倒排表等基本概念，全方位介绍了搜索引擎核心的索引技术。

第 6 章“搜索引擎的查询系统”介绍了查询系统的两个主要功能模块，即检索模块和摘要提取模块的工作方法和设计技巧，解开搜索引擎查询准确的奥秘所在。

第 7 章“搜索引擎的其他话题”通过回答有关搜索引擎的常见问题系统地介绍了搜索引擎各个系统的相互关系，并展望了搜索引擎未来的发展。

附录部分为搜索引擎系统全图，它将有助于理解搜索引擎各个系统的相互配合和协作关系。

如何阅读本书

本书中带★的章节中介绍了具有一定深度的理论知识，没有学过数学分析和概率

论等基础知识的读者可以浏览这些章节。而不需要深究推导过程，只需要了解一些基本的结论即可。

在本书的主要章节，即第 3 章至第 6 章中，第 5 章介绍的索引系统最为复杂，较难理解。读者在第 1 遍通读时，不需要深入细节。待阅读查询系统后，再回头阅读索引系统会有更大地帮助。对于一些难以理解的地方可以阅读章节结束部分给出的参考文献，深入理解这些难点问题。

非技术类读者可以忽略第 5 章中的大部分内容,以及其他章节中技术性强的部分，而集中精力关注搜索引擎的基本策略和方法。

附录部分为搜索引擎全部系统的架构图，它将有助于宏观上理解搜索引擎各个系统相互配合的过程。

关于本书作者

作者毕业于南京大学软件学院，获得软件工程硕士学位。曾经发表过多篇论文，获得 1 项国家专利。作者主要的兴趣方向包括数据挖掘、Web 挖掘、搜索引擎和软件工程等，目前在清华大学信息科学与技术国家实验室从事搜索引擎相关研究工作。

致谢

我首先要特别感谢我的妻子伍绍连，正是她的无私支持，使我能够全身心投入到写作中。在此书完成后，她通读了全书并提出了大量宝贵意见，使得本书增色不少。

感谢电子工业出版社计算机图书事业部孙学瑛女士，她是推动本书的完成最为关

键性的人物。她参与了此书创作的全过程，为笔者提供了有关图书市场的宝贵信息，使得本书更加面向读者。

感谢此书参考文献的全部作者、搜索引擎研究界的杰出科学家们，以及其他为此书提出宝贵技术意见的业界同行，正是你们杰出的成就和无私的帮助，才能使本书达到了一定的学术水平。

由于作者水平有限，书中不足及错误之处在所难免，敬请专家和读者给予批评指正。

梁 斌

2007 年 8 月

目　录

第一章　引言 1

时至今日，万维网迷宫般的复杂和魅力还在继续。因为它每天都在不断地产生、更新或消失各种各样的网页。其魅力依然，然而复杂不在。正是由于诞生了搜索引擎（search engine）这样伟大的技术，万维网复杂的局面才被打破。搜索引擎成为带领人们走出迷宫的灯塔，帮助千百万的网民便捷地找到重要的信息。

第一节　什么是搜索引擎 2

第二节　搜索引擎的发展简史 5

搜索引擎的发展历史 5

第三节　搜索引擎大事快览 15

第四节　国内著名搜索引擎 17

百度（www.baidu.com） 17

中搜（www.zhongsou.com） 18

天网（e.pku.edu.cn） 19

搜狗（www.sogou.com） 20

参考文献 21

第二章　搜索引擎概貌 23

万维网的发展迫切地要求一种快速、全面、准确且稳定可靠的信息查询方法，由于搜

索引擎满足了这 4 个需求，所以才奠定了其在科学技术上的高度。有人甚至把搜索引擎和操作系统并列为当今最为复杂的系统软件。

第一节 搜索引擎的主要需求 24
查得快 24
查得全 25
查得准 25
查得稳 27
第二节 搜索引擎的 4 大系统 28
搜索引擎的体系结构 28

第三章 搜索引擎的下载系统 31

在搜索引擎的 4 大系统中，第 1 个系统是下载系统。和航天运载火箭系统的动力系统一样，下载系统是搜索引擎大厦的基础。搜索的数据均来自于下载系统的工作，其工作方式巧妙、合理且强大。爬虫（也称为“Crawler”，中文译为“爬虫”，或者“蜘蛛”）是其中最华彩的乐章。让我们从爬虫开始，逐渐进入闪烁着奇异光芒的领地。

第一节 爬虫的发展历史 32
世界上第 1 个爬虫 32
爬虫的发展历程 33
第二节 万维网及其网页分析 34
蝴蝶结型的万维网 34
万维网的直径 37
万维网的规模及变化特征 39
网页的特征 39
第三节 有关爬虫的基本概念 41

爬虫......41
种子站点......41
URL......42
Backlinks......42
第四节　网页抓取原理......43
telnet 和 wget......43
从种子站点开始逐层抓取......44
不重复抓取策略......50
网页抓取优先策略......59
网页重访策略★......61
Robots 协议......67
其他应该注意的礼貌性问题......69
抓取提速策略（合作抓取策略）......70
第五节　网页库......77
第六节　下载系统回顾及未来发展......82
参考文献......84

第四章　搜索引擎的分析系统......86

搜索引擎的 4 大系统中的第 2 个系统是分析系统，分析系统主要完成的工作包括信息抽取、网页消重、中文分词和 PageRank 计算等。

第一节　知识准备......87
HTML 语言......87
锚文本（anchor text）......87
半结构化数据（Semi-structured data）......88

第二节　信息抽取及网页信息结构化……89
网页结构化的目标……89
建立 HTML 标签树……93
通过投票方法得到正文……98
网页结构化过程回顾……103
第三节　网页查重……105
网页查重技术发展历史……105
网页查重实现方法……107
第四节　中文分词……113
什么是中文分词……113
通过字典实现分词……114
通过统计学方法实现分词……120
第五节　PageRank……121
PageRank 的来由……121
PageRank 的基本想法……122
PageRank 的计算公式……124
PageRank 的计算方法 ★……129
第六节　分析系统结构图……134
参考文献……136

第五章　搜索引擎的索引系统……139

在搜索引擎的 4 大系统中，第 3 个系统称为“索引系统”。该系统就好像搜索引擎的数据大本营，在这里存储了并索引了数以亿计的网页。

第一节　知识准备……140

信息 140
索引 141
倒排索引、倒排表、临时倒排文件、最终倒排文件 141
其他概念 142

第二节　全文检索 143
全文检索 143

第三节　文档编号 146
编号的本质 146
文档编号的方法 147
游程编码 149

第四节　倒排索引 154
经典的倒排索引 154
正排索引（前向索引） 155
倒排索引 158

第五节　数据规模的估计 163
齐普夫法则 163
布尔检索模型下的索引规模估计★ 165

第六节　涉及存储规模的一些计算 170
正排表与倒排表的合并 170
多个临时倒排文件的归并 174
倒排索引分布式存储 179
倒排文件缓存 183
倒排索引词典统计信息的计算 183

第七节　倒排索引文件的创建过程 185

创建倒排表……185
计算统计信息……187
参考文献……189

第六章　搜索引擎的查询系统……191

在搜索引擎4大系统中，第4个系统称为“查询系统”。查询系统直接面对用户，在接收用户的查询请求后，通过检索、排序及摘要提取等计算，将计算结果组织成搜索结果页返回给用户。整个查询过程不仅要快，而且必须能够提供用户满意的查询结果。

第一节　知识准备……192
什么是信息熵……192
检索和查询的区别……196
检索词和查询词的区别……196
自动文本摘要(Automatic Text Summarization)……197
第二节　网页信息检索……198
早期的检索模型……198
向量空间模型（Vector Space Models）……201
关键词权重的量化方法 TF/IDF★……207
搜索引擎采用的检索模型……213
多文档列表求交计算……215
检索结果排序……222
堆排序……223
第三节　中文自动摘要……230
自动摘要的发展历史……230
自动摘要的含义和实现……231

第四节　生成搜索结果页......239
生成搜索结果页......239
第五节　搜索结果页的缓存......242
搜索结果页的缓存......242
第六节　推测用户查询意图......245
查询分类......245
推测信息类、事物类的查询意图......247
第七节　查询系统的当前热点和发展方向......249
查询系统的当前热点......249
参考文献......250

第七章　搜索引擎的其他话题 252

本书初步介绍了搜索引擎 4 大系统各自的运作原理，以及相互配合的关系。本节通过回答一些有关搜索引擎的基本问题，以从宏观上更好地理解和认识搜索引擎。

第一节　搜索引擎问与答......253
为什么搜索引擎的搜索速度这么快......253
为什么搜索引擎能够返回那么多的查询结果......255
为什么搜索引擎总能返回最想要的结果......256
搜索引擎如何大规模存储网页的......257
什么是 SEO......259
什么是元搜索引擎......260
搜索引擎认为的作弊行为是哪些......261
如何进一步学习和了解搜索引擎发展的最新成果......262
第二节　搜索引擎未来的发展......265

新兴的搜索产品 265
搜索技术的未来 268
参考文献 270
附录 A　搜索引擎系统结构全观图 271

第一章

引言

01 第一节 什么是搜索引擎

随着互联网的蓬勃发展，建立在互联网之上的各种应用也层出不穷，其中最为成功的莫过于万维网（WWW）。万维网被称为“网中之网”，是互联网上最受欢迎的服务之一。它运用超文本技术为人们访问信息资源提供了巨大的方便，但也以非线性组织的构建方式使人们在信息海洋中彷徨。奥地利的鲁施在 1994 年接触万维网，并在其作品《令人吃惊的万维网》（aMAZEingweb）中表达了对万维网的感受：它有那么可观的潜力，却又是经常使探索者丧失方向的迷宫（MAZE）。

时至今日，万维网迷宫般的复杂和魅力还在继续。因为它每天都在不断地产生、更新或消失各种各样的网页。其魅力依然，然而复杂不在。正是由于诞生了搜索引擎（search engine）这样伟大的技术，万维网复杂的局面才被打破。搜索引擎成为带领人们走出迷宫的灯塔，帮助千百万的网民便捷地找到重要的信息。

WordNet 上的解释为搜索引擎是一种用来在计算机网络，特别是在万维网上检索各种文件的计算机程序。从本质上讲，如果将搜索引擎的搜索结果看做一种动态网页，那么这种动态网页通过提交的检索关键词聚合了各种重要、有价值并与关键词相关的

网页。因此，与其说搜索引擎是一个查询系统，不如说它是一个用户定义的信息聚合系统。通过用户输入的查询关键词，搜索引擎推测用户的查询意图，然后快速地返回相关的查询结果供用户选择。

对搜索引擎的理解也经历了一个漫长的过程，从早期的目录式搜索，到今天的全文搜索，人们对搜索引擎的认识也在不断地加深。今天，公认的搜索引擎有如下 3 种服务方式。

（1） 目录式搜索引擎

在万维网出现早期，信息检索通常通过人工发现信息，依靠编辑人员的知识进行甄别，并在此基础上进行分类。用户可以在这个分类结构中浏览，这就是我们熟知的目录检索系统。这种搜索引擎最有名的是早期的雅虎(Yahoo),以及国内的搜狐(Sohu)。该类搜索引擎因为加入了人的智能，所以信息准确且导航质量较高。目录式搜索引擎的特点是检索的目标结果是网站，可以看做是网站的黄页查询；不足是数据量有限、更新不及时，并且人工维护成本较高。

（2） 全文搜索引擎

全文搜索引擎是针对万维网所有网页进行全文检索的搜索引擎，由下载系统以某种策略自动地在万维网上搜集和发现信息，由索引系统为搜集到的信息建立索引，由查询系统根据用户的查询输入检索索引库，并将查询结果返回给用户。服务方式是面向网页的全文检索服务，该类搜索引擎的优点是信息量大、更新及时，并且毋需人工干预；缺点是返回信息过多，有很多无关信息，

用户必须从结果中筛选。其代表是 Google 及百度等第二代商用搜索引擎。

（3） 元搜索引擎（Meta-Search Engine）

这类搜索引擎没有自己的数据，而是将用户的查询请求同时向多个搜索引擎递交。然后将返回的结果进行重复排除及重新排序等处理后，作为自己的结果返回给用户。服务方式为面向网页的全文检索。这类搜索引擎的优点是返回结果的信息量大；缺点是不能够充分使用原搜索引擎的功能，用户需要做更多的筛选，其代表是 WebCrawler。

上述 3 种搜索引擎共经历了不到 20 年的发展历程，然而就是在这短短的不到 20 年的时间里，在一代代的搜索技术精英不断地努力下，成就了今天伟大而卓越的搜索引擎技术。其中的很多技术成果也用到了其他领域，创造了巨大的价值。

接下来，我们将回到搜索发展历史的长河，追溯搜索发展的历史。

第二节 搜索引擎的发展简史 01

读书笔记

万维网信息爆炸和计算机软硬件水平的发展成为搜索引擎的诞生、发展并壮大的最大推动力，下面我们将一起回顾在搜索引擎发展史上做出过突出贡献的技术精英及其杰出成就。（[Heting Chu et al 2002]，[李晓明 2004]，[唐铭杰 2001]，[W3 SEARCH ENGINES]）

搜索引擎的发展历史

1990 年以前，没有任何人能搜索万维网。

所有搜索引擎的祖先是 1990 年由 Montreal 的 McGill University 学生 AlanEmtage、Peter Deutsch 和 Bill Wheelan 发明的 Archie（Archie FAQ）。虽然当时 World WideWeb 尚未出现，但网络中的文件传输还是相当频繁的。由于大量的文件散布在各个分散的 FTP 主机中，查询起来非常不便。因此 Alan Emtage 等想到了开发一个可以用文件名查找文件的系统，于是便有了 Archie。它是第 1 个自动索引万维网上匿名 FTP 网站文件的程序，

但还不是真正的网络搜索引擎（Web Search Engine）。Archie 是一个可搜索的 FTP 文件名列表，用户必须输入精确的文件名搜索，然后 Archie 会告诉用户哪一个 FTP 地址可以下载该文件。

由于 Archie 深受欢迎，因此受其启发，Nevada System Computing Services 大学于 1993 年开发了一个 Gopher（Gopher FAQ）搜索工具 Veronica（Veronica FAQ）。Jughead 是后来另一个 Gopher 搜索工具。

Robot（机器人）一词对编程者有特殊的意义，Computer Robot 是指某个能以人类无法达到的速度不断重复执行某项任务的自动程序。由于专门用于检索信息的 Robot 程序如同蜘蛛（spider）一样在网络间爬来爬去，因此搜索引擎的 Robot 程序被称为"Spider（Spider FAQ）程序"。世界上第 1 个 Spider 程序是 MIT MatthewGray 的 World Wide Web Wanderer，用于追踪万维网的发展规模。刚开始它只用来统计万维网上的服务器数量，后来则发展为也能够捕获网址（URL）。

与 Wanderer 相对应，1993 年 10 月，Martijn Koster 创建了 ALIWEB（MartijnKoster Annouces the Availability of Aliweb）。它相当于 Archie 的 HTTP 版本，并且不使用网络搜寻 Robot。如果网站主管们希望自己的网页被 ALIWEB 收录，需要自己提交每一个网页的简介索引信息，类似于后来大家所熟知的 Yahoo。

随着万维网的迅速发展，使得检索所有新出现的网页变得越来越困难。因此在 Wanderer 基础上，一些编程者对传统的 Spider 程序工作原理做了些改进。其设想是既然所有网页都可能有连向其他网站的链接，那么从一个网站开始跟踪所有网页上的所有链

接，就有可能检索整个万维网。到 1993 年年底，一些基于此原理的搜索引擎开始纷纷涌现，其中最负盛名的是 Scotland 的 JumpStation、Colorado 大学 Oliver McBryan 的 The World Wide Web Worm（FirstMention of McBryan's World Wide Web Worm）和 NASA 的 Repository-BasedSoftware Engineering（RBSE）spider。JumpStation 和 WWW Worm 只是以搜索工具在数据库中找到匹配信息的先后次序排列搜索结果，因此毫无信息关联度可言。而 RBSE 是第 1 个索引 Html 文件正文的搜索引擎，也是第 1 个在搜索结果排列中引入关键字串匹配程度概念的引擎。

1. Excite

Excite（http://www.excite.com）由当时还是一个以车库为办公室的软件公司 Architext Software 研发，1996 年其宣传已经能够索引 150 万的 Web 网页，而且每周更新一次索引。它不仅能够提供关键词查询，还能够提供概念查询。2002 年 5 月被 Infospace 收购的 Excite 停止自己的搜索引擎，改用元搜索引擎 Dogpile。

2. Yahoo

Yahoo（http://www.yahoo.com），1994 年 4 月，Stanford University 的两名博士生，美籍华人 Jerry Yang（杨致远）和 David Filo 共同创办了 Yahoo（Jerry Yang Alerts a Usenet group to the YahooDatabase，1996 年的 Yahoo）。随着访问量和收录链接数的增长，Yahoo 目录开始支持简单的数据库搜索。因为 Yahoo!的数据是手工输入的，所以不能真正被归为搜索引擎，事实上只是一个可搜索的目录。Wanderer 只抓取 URL，但 URL 信息含量太小，

读书笔记

很多信息难以单靠 URL 说清楚，搜索效率很低。Yahoo!中收录的网站，因为都附有简介信息，所以搜索效率明显提高（注：Yahoo 以后陆续使用 Altavista、Inktomi、Google 提供搜索引擎服务）

3. WebCrawler

1994 年年初，Washington 大学的 CS 学生 Brian Pinkerton 开始其小项目 WebCrawler（Brian Pinkerton Announces the Availability of Webcrawler）。1994 年 4 月 20 日，WebCrawler 正式亮相，当时仅包含来自 6 000 台服务器的内容，它是万维网上第 1 个支持搜索文件全部文字的全文搜索引擎。在其之前，用户只能通过 URL 和摘要搜索，摘要一般来自人工评论或程序自动取正文的前 100 个字（后来 Webcrawler 陆续被 AOL 和 Excite 收购，现在和 Excite 一样改用元搜索引擎 Dogpile）。

4. Lycos

Lycos（http://www.lycos.com），是狼蛛（wolf spider）的拉丁文前 5 个字母构成，诞生于号称美国 4 大计算机高校的 Carnegie Mello 大学。不久被美国在线（America Online）购买，成为 Lycos 公司，创始人 Michael Mauldin 继续在 Lycos 公司工作，而 Lycos 也继续为万维网社区提供免费的查询服务。Lycos 是第 1 个在搜索结果中使用了网页自动摘要，而最大的优势还是它远胜过其他搜索引擎的数据量。1996 年 1 月，Lycos 已经索引了 95%的 Web 资源（包括 1 900 万个独立 URL），因此成为当时屈指可数的大型搜索引擎。然而它还是没有能够索引网页的全文信息（the full text of a Web page），仅仅能够检索标题和正文的前 20 行或者前 20%

的正文，Lycos 的竞争对手在这个明显的弱点上不断大做文章。

5. Infoseek

Infoseek(Steve Kirsch Announces Free Demos Of the Infoseek Search Engine) 是另一个重要的搜索引擎，虽然公司声称于 1994 年 1 月已创立，但直到年底其搜索引擎才与公众见面。起初，Infoseek 只是一个不起眼的搜索引擎，它沿袭 Yahoo!和 Lycos 的概念，并没有什么独特的革新。但是其发展史和后来受到的众口称赞证明，第 1 个登台并不总是很重要。Infoseek 友善的用户界面及大量附加服务（such as UPS tracking, News, a directory, and the like）使其声望日隆。而 1995 年 12 月与 Netscape 达成的战略性协议，使它成为一个强势搜索引擎。当用户点击 Netscape 浏览器上的搜索按钮时，弹出 Infoseek 的搜索服务，而此前由 Yahoo 提供该服务（Infoseek 后来曾以相关性闻名，2001 年 2 月，它停止了自己的搜索引擎，开始改用 Overture 的搜索结果）。

6. Alta Vista

Alta Vista(http://www.altavista.digital.com)。1995 年的夏天，在位于加利福利亚 Palo Alto 的数字研究实验室里，Alta Vista 的研究计划被正式启动。1995 年的 12 月 15 日，Alta Vista 正式发布。到 1996 年，Alta Vista 已经索引了 1 600 万的网页全文，其宣称每天抓取 250 万的网页，每小时索引 1 GB 容量的文本。Alta Vista 最大的贡献在于支持了布尔查询、单句查询（phrase searching）及特定域查询（例如标题查询和 URL 查询等），而且支持查找网页的 Backlinks，即查询一个网页被哪些网页所指向。

读书笔记

Alta Vista 的第 1 个支持用户自己向网页索引库提交或删除 URL 的搜索引擎，并能在 24 小时内上线。

7. HotBot

随后到来的是 HotBot，1995 年 9 月 26 日，加州伯克利分校 CS 助教 EricBrewer 和博士生 Paul Gauthier 创立了 Inktomi（UC Berkeley Announces Inktomi）。1996 年 5 月 20 日，Inktomi 公司成立，强大的 HotBot 出现在世人面前。它声称每天能抓取 1 000 万页以上的索引，所以有远超过其他搜索引擎的新内容。HotBot 也大量运用 cookie 保存用户的个人搜索喜好设置（Hotbot 曾是随后几年最受欢迎的搜索引擎之一，后被 Lycos 收购）。

8. Northernlight

Northernlight 公司于 1995 年 9 月成立于马萨诸塞州剑桥，1997 年 8 月，Northernlight 搜索引擎正式现身。作为曾拥有最大数据库的搜索引擎之一，它没有 Stop Words，但有出色的 Current News、7 100 多个出版物组成的 SpecialCollection、良好的高级搜索语法，以及第 1 个支持对搜索结果进行简单的自动分类（2002 年 1 月 16 日，Northernlight 公共搜索引擎关闭，随后被 divine 收购。但在 Nlresearch，选中“World Wide Web only”，仍可使用 Northernlight 搜索引擎）。

9. Sohu（http://www.sohu.com）

1996年中国出现了类似雅虎（Yahoo）的网站“搜狐”，在许多场合，也称 Yahoo!之类的门户网站提供的信息查找功能为“搜索引擎”。但从技术上讲，这样的门户中提供的搜索服务和前述搜索引擎是很不同的。这样的门户依赖的是人工整理的网站分类目录，一方面，用户可以直接沿着目录导航，定位到自己所关心的信息；另一方面，用户也可以提交查询词，让系统将其直接引导到和该查询词最匹配的网站。可以说这种是网站级别的查询，而非网页级别的查询，也有人称这种服务为网站黄页服务。

10. Google（http://www.google.com）

1998年10月之前，Google只是Stanford大学的一个小项目BackRub。1995年博士生Larry Page开始学习搜索引擎设计，并于1997年9月15日注册了google.com的域名。1997年年底，在Sergey Brin、Scott Hassan和Alan Steremberg的共同参与下，BachRub开始提供Demo。1999年2月，Google完成了从Alpha版到Beta版的蜕变。Google公司则把1998年9月27日认做自己的生日。Google在Pagerank、动态摘要、网页快照、DailyRefresh、多文档格式支持、地图股票、词典寻人等集成搜索、多语言支持，以及用户界面等功能上的革新，与Altavista一样，再一次永远改变了搜索引擎的定义。在2000年以前，Google虽然以搜索准确性备受赞誉，但因为数据库不如其他搜索引擎大，并缺乏高级搜索语法，所以使用价值不是很高，推广并不快。直到2000年数据库升级后，又借被Yahoo选为搜索引擎的东风，才一飞冲天。

11. Fast（Alltheweb）

Fast (Alltheweb) 公司创立于 1997 年，是挪威科技大学 (NTNU) 学术研究的副产品。1999 年 5 月，该公司发布了自己的搜索引擎 AllTheWeb。Fast 创立的目标是做世界上最大和最快的搜索引擎，几年来庶几近之。Fast (Alltheweb) 的网页搜索可利用 ODP 自动分类，支持 Flash 和 pdf 搜索，支持多语言搜索，还提供新闻、图像、视频、MP3 和 FTP 搜索，拥有极其强大的高级搜索功能。

12. Teoma

Teoma 起源于 1998 年 Rutgers 大学的一个项目，Apostolos Gerasoulis 教授带领华裔 Tao Yang 教授等人于新泽西 Piscataway 创立 Teoma，2001 年春初次登场。2001 年 9 月被提问式搜索引擎 Ask Jeeves 收购，2002 年 4 月再次发布。Teoma 的数据库目前仍偏小，但有两个出彩的功能。即支持类似自动分类的 Refine，同时提供专业链接目录的 Resources。

13. Wisenut

Wisenut 由韩裔 Yeogirl Yun 创立，2001 年春季发布 Beta 版，2001 年 9 月 5 日发布正式版，2002 年 4 月被分类目录提供商 looksmart 收购。Wisenut 也有两个出彩的功能，即包含类似自动分类和相关检索词的 WiseGuide，以及预览搜索结果的 Sneak-a-Peek。

14. Gigablast

Gigablast 由前 Infoseek 工程师 Matt Wells 创立，2002 年 3 月展示了 pre-beta 版，2002 年 7 月 21 日发布 Beta 版。Gigablast 的数据库目前仍偏小，但也提供网页快照。一个特色功能是即时索引网页，你的网页刚提交，它就能搜索（这个 spammers 的肉包子功能暂已关闭）。

15. Openfind

Openfind 创立于 1998 年 1 月，其技术源自台湾中正大学吴升教授所领导的 GAIS 实验室。它起先只做中文搜索引擎，曾经是最好的中文搜索引擎。在其鼎盛时期，同时为 3 大著名门户新浪、奇摩和雅虎提供中文搜索引擎，但 2000 年后市场逐渐被 Baidu 和 Google 瓜分。2002 年 6 月，Openfind 重新发布基于 GAIS30Project 的 Openfind 搜索引擎 Beta 版，推出多元排序（PolyRankTM），宣布累计抓取网页 35 亿。并且开始进入英文搜索领域，此后技术升级明显加快。

16. 北大天网

北大天网是国家“九五”重点科技攻关项目“中文编码和分布式中英文信息发现”的研究成果，由北大计算机系网络与分布式系统研究室开发，北京大学计算机系的项目组在陈葆珏教授的主持下于 1997 年 10 月在 CERNET 上推出了天网搜索 1.0 版本。该系统在这几年里不断发展，目前已成为中国最大的公益性搜索引擎（http://e.pku.edu.cn）。2000 年初成立天网搜索引擎新课题

读书笔记

组，由国家 973 重点基础研究发展规划项目基金资助开发。它收录网页约 6 000 万个，利用教育网优势，有强大的 FTP 搜索功能。

17. Baidu

2000 年 1 月，超链分析专利发明人、前 Infoseek 资深工程师李彦宏与好友徐勇（加州伯克利分校博士）在北京中关村创立了百度（Baidu）公司。2001 年 8 月，发布 Baidu.com 搜索引擎 Beta 版（此前 Baidu 只为其他门户网站搜狐、新浪和 Tom 等提供搜索引擎）。2001 年 10 月 22 日，正式发布 Baidu 搜索引擎。Baidu 虽然只提供中文搜索，但目前收录中文网页超过 9 000 万个，可能是最大的中文数据库。Baidu 搜索引擎的其他特色包括网页快照、网页预览/预览全部网页、相关搜索词、错别字纠正提示、新闻搜索、Flash 搜索，以及信息快递搜索等。2002 年 3 月，闪电计划（Blitzen Project）开始后，其技术升级明显加快。

第三节 搜索引擎大事快览 01

2002 年 12 月 24 日，雅虎称公司同意以大约 2.35 亿美元的价格收购搜索软件公司 Inktomi。

2003 年 1 月 18 日，Google 收购博客网站 Blogger.com 开发团队——网上出版软件开发商 Pyra Labs。

2003 年 2 月 19 日，Overture 服务公司表示，计划以 1.4 亿美元现金加股票从 CMGI 公司手中收购门户网站 AtaVista。

2003 年 2 月 26 日，Overture 同意以 1 亿美元收购位于挪威的 Fast Search andTransfer 公司的网络搜索部门。

2003 年 4 月 15 日，新浪与中国搜索联盟结成战略同盟。至此，中国已有数百家网站结成搜索联盟，以迎接国际巨头 Google 挺进国内市场后的巨大压力。

2003 年 4 月 21 日，第 2 大万维网搜索引擎提供商 Ask Jeeves 公司宣布对其 Ask.com 网站进行升级。Ask Jeeves 是仅次于 Google 的第 2 大搜索引擎，也是万维网上第 5 大搜索基地（Google、雅虎、微软、AOL 及 Askjeeves）。

2003 年 6 月 18 日，微软公司表示其正在加大研发新型万维网搜索引擎技术的力度，包括对一款功能更先进的技术原型进行测试。

2003 年 7 月 12 日，从加利福里亚传来消息，Google 即将把总部从 BayshoreParkway 搬迁至半里之遥的一个有 4 栋楼房的复式结构建筑中，这个建筑是由鼎鼎大名但目前却陷入困境的硅谷图像（Silicon Graphics）公司腾出来的。大卫.奎恩（David Krane）证实了这个消息，并解释说，这样能让公司现有的 800 多员工更好地分工合作和管理。

2003 年 7 月 15 日，全球最大的互联网公司雅虎宣布，以 16.3 亿美元收购在网络搜索服务上的竞争对手 Overture 公司，以期在同 Google 的竞争中取得优势。

2004 年 8 月 3 日，搜狐公司正式推出搜狗搜索引擎，使得中文搜索引擎行业进入了一个空间繁荣的历史局面。

2005 年 9 月，Google 正式进入中国市场。

搜索引擎的出现虽然只有 20 年左右的历史，但在 Web 上已经有了确定不移的地位。据 CNNIC 统计，它已经成为继电子邮件之后的第 2 大 Web 应用，越来越多的网民在日常网络生活中都会在搜索引擎的帮助下寻找感兴趣的各种信息。

时至今日，虽然搜索引擎的基本工作原理和技术已经趋于成熟，但是在质量、性能和服务方式等方面的提高空间依然很大，与之相关的研究成果层出不穷，因此搜索引擎相关的话题也是每年 WWW、SIGIR 及 SIGMOD 等重大学术会议的重要议题。

第四节 国内著名搜索引擎 01

尽管国内的搜索引擎发展起步相对较晚，然而国内的搜索引擎却发展得异常繁荣，成为世界搜索引擎行业中一道亮丽的风景线。由于本书立足介绍中文搜索引擎，因此本节介绍一些国内著名的搜索引擎。

百度（www.baidu.com）

百度的主页如图 1-1 所示。



图 1-1　www.baidu.com

2000 年 1 月 1 日，公司创始人李彦宏和徐勇携 120 万美元风险投资，从美国硅谷回国。于北京中关村创建了百度公司目前是全球最大的中文搜索引擎。

2000 年 5 月，百度首次为门户网站——硅谷动力提供搜索技术服务。之后迅速占领中国搜索引擎市场，成为最主要的搜索技术提供商。

2001 年 8 月，百度发布 Baidu.com 搜索引擎 Beta 版。从后台服务转向独立提供搜索服务，并且在中国首创了竞价排名商业模式。2001 年 10 月 22 日，正式发布 Baidu 搜索引擎。

2005 年 8 月 5 日，百度在美国纳斯达克上市，成为 2005 年全球资本市场上最为引人注目的上市公司之一。

百度主要提供中文信息检索，搜索范围涵盖中国内地，香港、台湾和澳门地区，新加坡等华语地区，以及北美和欧洲的部分站点。百度的产品线齐全，除了网页搜索以外，比较著名的有包括百度贴吧、百度知道和音乐搜索等。正是由于百度在技术上的领先，所以奠定了其在中国互联网高科技公司的优势地位，成为中国互联网信息服务的旗帜之一。

中搜（www.zhongsou.com）

中搜的主页如图 1-2 所示。



图 1-2　www.zhongsou.com

2002 年，慧聪搜索正式进入中文搜索引擎市场。

2003 年 8 月 24 日，慧聪搜索（现中国搜索）正式推出第 3 代智能中文搜索引擎。

2003 年 12 月 23 日，慧聪搜索正式独立运作，成立中国搜索，陈沛出任 CEO，同时推出中国搜索新闻中心。

2004 年 2 月 26 日，中国互联网新闻中心和 IDG 集团注资中国搜索成立合资公司，并提出“个人门户时代”的创新理念。

中搜不断创新，走自己的搜索道路，在个人门户的探索上取得了一定的成绩。

天网（e.pku.edu.cn）

天网的主页如图 1-3 所示。



图 1-3　e.pku.edu.cn

天网中英文搜索引擎是国家“九五”重点科技攻关项目“中文编码和分布式中英文信息发现”的研究成果。

1997 年 10 月 29 日，天网正式在 CERNET 上向广大互联网用户提供 Web 信息导航服务。

2000 年年初，新成立的“天网”搜索引擎课题组获得国家 973 重点基础研究发展规划项目基金资助。

在李晓明教授的带领下成功开发了中国第 1 个搜索引擎，并完成了大量高质量的学术论文。此外天网搜索实验室也成为了中国搜索引擎工程师的摇篮，为中国搜索行业输送了大量高科技人才。

搜狗（www.sogou.com）

搜狗的主页如图 1-4 所示。

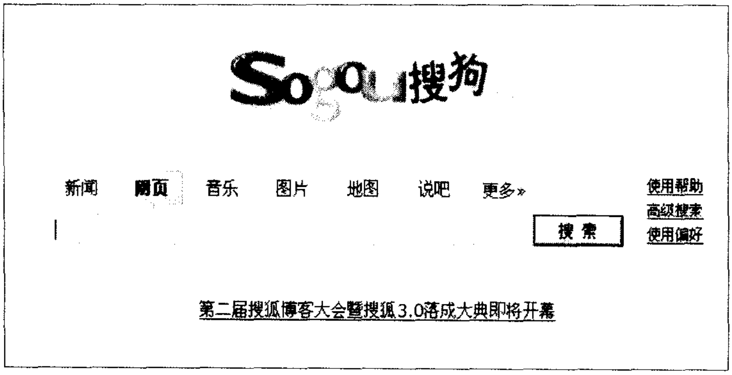


图 1-4 www.sogou.com

搜狗是搜狐公司于 2004 年 8 月 3 日推出的全球首个第 3 代互动式中文搜索引擎。

2007 年 1 月 1 日，搜狐公司正式推出网页搜索 3.0 版本，全面升级的搜狗网页搜索 3.0 凭借自主研发的服务器集群并行抓取技术，成为全球首个中文网页收录量达到 100 亿的搜索引擎。

“更懂网络、更懂中文搜索”成为搜狗不断技术创新的动力，特别是 2006 年 6 月 5 日推出的搜狗中文输入法成为搜狗技术创新的集中代表，成为商用搜索引擎领域的后起之秀。

参考文献

[1] [Heting Chu et al 2002] Heting Chu, Marilyn Rosenthal Search Engines for the World Wide Web: A Comparative

Study and Evaluation Methodology ASIS 1996 Annual Conference Proceedings October 19-24,196

[2] [baidu about]http://www.baidu.com/about/ index.html

[3] [sogou about]http://www.sogou.com/docs/ about.htm

[4] [tianwang about]http://e.pku.edu.cn/gbabout project.htm

[5] [zhongsou about]http://www.zhongsou.com/ jjzs/gaikuang/ index.html

[6] [W3 SEARCH ENGINES] http://vlib.iue.it/ history/search/

[7] [李晓明 2004] 李晓明，闫宏飞，and 王继民，搜索引擎-原理、技术与系统：科学出版社

[8] [唐铭杰 2001]唐铭杰.论搜索引擎的发展概况及发展趋势[J].情报杂志,2001,20（5）:70-71

第二章

搜索引擎概貌

02 第一节 搜索引擎的主要需求

随着万维网信息爆炸性地飞速增长，传统的查询方法无法为网民提供有效的服务。万维网的发展迫切地要求一种快速、全面、准确且稳定可靠的信息查询方法，由于搜索引擎满足了这 4 个需求，所以才奠定了其在科学技术上的高度。有人甚至把搜索引擎和操作系统并列为当今最为复杂的系统软件，下面我们将对搜索引擎中最为主要的 4 个需求，同样也是搜索引擎的主要特点加以说明。

查得快

一方面，随着信息化社会的到来，信息可以说是无处不在，人们的日常生活离不开这些有价值的信息；另一方面，人们的生活节奏也在不断地加快，人们应该能够平等地获得这些公众信息。这就要求搜索引擎必须能够存储这些无处不在的信息并且能够快速地进行信息查询，满足网民的查询需求。

有调查表明，当今公开的搜索引擎的查询速度都在秒这个量

级以下，商用搜索引擎的查询速度达到毫秒级，并且能够支持大规模用户的同时访问。

影响速度的原因很多，例如索引库的效率、分布查询的处理能力和查询缓存的命中率等，这些将在第四章和第五章中详细介绍。

查得全

在传统信息检索（information retrieval）中，应用查全率（Recall）作为衡量检索是否全面的度量指标（本章不区分查询和检索的区别），查全率是检索出的相关网页数和所有的相关网页数的比率。例如在搜索引擎中查询“XML”，如果世界上包含“XML”这个关键词的网页数为 M，而实际该搜索引擎检索出这 M 条中的 N 条网页，那么查全率为 N/M×100%。

是否能查得全，主要取决于网页索引库的大小。从这一点上讲，索引的网页数越多，越有助于提高查全率。

查得准

在传统信息检索中，应用查准率（Precision）作为衡量检索是否准确的指标，查准率是检索出的相关文档数与检索出的文档总数的比率。例如在搜索引擎中查询“XML”，在实际检索出的网页数 N 中，只有 P 个网页是与查询“XML”相关的，那么查准率为 P/N×100%。

通过图 2-1，可以全面理解查全率和查准率的关系。

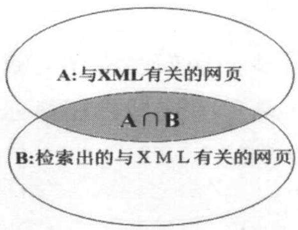


图 2-1　查全率和查准率的关系

查全率=$\frac{|A\cap B|}{|A|}$，其中对集合取$|\,|$运算的结果表示集合的数量。

查准率=$\frac{|A\cap B|}{|B|}$。

在搜索引擎这种特殊的检索实践中，查全率往往是不重要的。衡量的意义也不大，因为没有一个用户会把所有与查询相关的网页都浏览一遍。一般情况下，用户最为关注的仅仅为搜索结果中的前几条。而查准率在很大程度上决定了搜索的质量，在前 10 条搜索结果（搜索结果首页）中满足用户的查询目的是搜索引擎查准率的主要体现。

是否能查得准，主要取决于网页排序。常见的有 PageRank 等排序方法，在第四章中将介绍这方面的内容。

读书笔记

查得稳

毫无疑问，搜索引擎必须是一个能够长期并稳定地提供服务的系统，因此系统的稳定运行是很重要的需求。特别是商用搜索引擎，其稳定性被提高到了相当的高度。在任何情况下可以牺牲检索质量和检索速度，但必须能够提供持续的信息检索服务。

对于搜索引擎来说，查询来自四面八方，查询词也千差万别，同时进行的查询量也非常巨大。稳定地满足这些查询需要，需要在系统的结构上做出权衡，在文件存储方式、查询系统和索引系统设计等方面都需要考虑稳定性的因素。

02 第二节 搜索引擎的4大系统

无论工业界还是学术界，搜索引擎一致地被认为分为下载、分析、索引和查询4大系统。这4大系统相互配合，共同实现了搜索引擎的快、全、准、稳的4大需求。

搜索引擎的体系结构

搜索引擎的结构清晰，分工明确。按照各自的功能划分，分为以下4大系统。

（1）下载系统。

（2）分析系统。

（3）索引系统。

（4）查询（检索）系统。

其中下载系统负责从万维网上下载各种类型的网页，并且保持对万维网变化的同步，将在第三章中详细介绍。

分析系统负责抽取下载系统得到的网页数据，并进行PageRank和分词计算，将在第四章中详细介绍。

索引系统负责将分析系统处理后的网页对象索引入库，将在第五章中详细介绍。

查询系统负责分析用户提交的查询请求，然后从索引库中检索出相关网页并将网页排序后，以查询结果的形式返回给用户，将在第六章中详细介绍。

搜索引擎详细的简要结构图如图 2-2 所示。

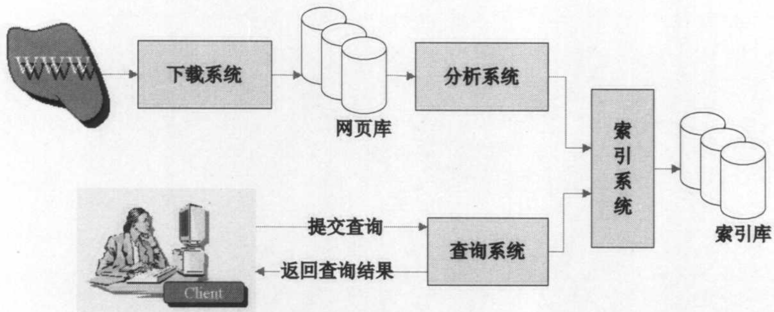


图 2-2　搜索引擎的简要结构图

从整体上看，下载系统、分析系统和索引系统组成了搜索引擎的数据制作部分，被称为是“离线部分”(offline part)；查询系统为搜索引擎的数据服务部分，要求快速响应，因此被称为“在

线部分"(online part)。按照离线和在线划分，搜索引擎又可分为在线系统和离线系统。其中在线系统需要毫秒级的访问速度，而离线系统中有些需要长达几周的时间才能计算完毕。

从细节上看，网页从开始到最后都是网页。而在搜索引擎的内部会有两种不同的形式，一种以网页库的方式存储；一种成为网页对象被存储在索引库中。搜索引擎的主要数据来自网页，网页处理能力是搜索引擎面对的主要挑战，下载系统和索引系统分别会进行一些关于数据存储的相关知识的介绍。

一个网页在万维网（WWW）中诞生，然后被下载系统下载。进而被分析并索引入库，最后因为该网页包含的一个关键词被检索而进入用户（Client）的大脑。这样一个奇妙的旅行都经历了哪些细节？各个系统内部如何工作？搜索引擎的全部画卷将在接下来的 4 章中按照这个顺序一一展开。

第三章

搜索引擎的下载系统

03 第一节 爬虫的发展历史

在搜索引擎的 4 大系统中，第 1 个系统是下载系统。和航天运载火箭系统的动力系统一样，下载系统是搜索引擎大厦的基础。搜索的数据均来自于下载系统的工作，其工作方式巧妙、合理且强大。爬虫（也称为“Crawler”，中文译为“爬虫”，或者“蜘蛛”）是其中最华彩的乐章。让我们从爬虫开始，逐渐进入闪烁着奇异光芒的领地。

世界上第 1 个爬虫

爬虫是一种自动抓取万维网网页信息的机器人。世界上第 1 个网络爬虫由麻省理工学院（MIT）的学生马休·格雷（Matthew Gray）在 1993 年写成，并命为“万维网漫游者”）。尽管其编写目的不是为了做搜索引擎，但是正是这革命性的创新，为搜索引擎的发展和今天的广泛应用奠定了坚实的基础。

爬虫的发展历程

现代搜索引擎的思路源于 Wanderer，不少人改进了 Matthew Grey 的蜘蛛程序。1994 年 7 月，Michael Mauldin 将 John Leavitt 的蜘蛛程序接入到其索引程序中，创建了目前为人熟知的 Lycos（http://www.lycos.com/）。其后无数的搜索引擎促使爬虫越写越复杂，并逐渐向多策略、负载均衡及大规模增量抓取等方向发展。爬虫的工作成果使得搜索引擎能够检索几乎全部的万维网网页，甚至被删除的网页也可以通过一个称之为“网页快照”的功能访问。

前人的辉煌成就令人赞叹不已，那么爬虫是怎么实现这些功能的呢？为什么说它巧妙、合理且强大呢？让我们首先从爬虫开始入手，深入理解搜索引擎的下载系统。

03 第二节 万维网及其网页分析

在深入爬虫领地之前，不妨把爬虫看做是劳动者，把机器和网络带宽资源看做是劳动资料，把万维网上的网页看做是劳动对象。因此，深入并透彻地理解劳动对象才能更加科学地理解劳动者的工作原理。

蝴蝶结型的万维网

首先从万维网的静态结构入手，如果将万维网定义为一个相互连通的连通图，网页为结点，链接（link）为边，那么任意一个网页可能被其他网页链接，这种链接称为"反向链接"（back link）。这个网页也可能链接到其他网页，这种链接称为"正向链接"（简称"链接"）。

Broder 通过一个有趣的 Random-start BFS（Breadth-First Search）实验随机取得 570 个网页作为开始结点，依次逐个地试验这 570 个网页。首先采用正向（forward direction）宽度优先遍历（宽度优先遍历的方法将在后面检索），即按照网页到网页的链

接进行遍历。然后采用反向（backward）宽度优先遍历，即按照网页到网页反向链接进行遍历。例如网页 A 包含网页 B 的链接，则从网页 A 开始，正向遍历可以遍历到网页 B；从网页 B 开始，反向遍历可以遍历到网页 A。

通过实验发现，无论是正向遍历还是反向遍历，表现出来的是截然不同的遍历效果。要么是在遍历到很小的结果集（90%的情况下探测到的集合大小不超过 90 个结点，极端情况下也只有几万个网页而已）后遍历终止；要么是爆炸性地遍历到了大约 1 亿个网页，但是没有探测到全部的 18 600 万个结点。此外，对于一部分的起始点，无论是正向，还是反向遍历都能够探测出大约 1 亿个网页，这部分起始点属于后面提到的 SCC 部分。通过实验数据，Broder 得出这样一个结论，即万维网具有蝴蝶结型（bow tie）结构，如图 3-1 所示。

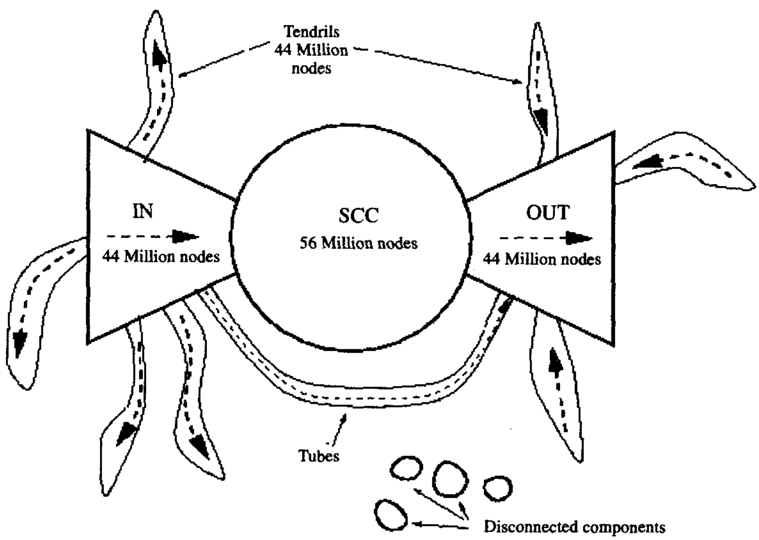


图 3-1　万维网的蝴蝶结型结构

其中的网页分为如下 4 种类型，各约占 1/4。

（1）蝴蝶结的中部（SCC，Strongly Connected Component）。

这种类型的网页彼此相连，任意去掉有限个网页，不会影响其连通度。Random-start BFS 遍历实验的起始点选择该部分网页，采用正向遍历的方法，从统计的角度上看可以遍历占全部网页 3/4 的网页数；采用反向遍历的方法也可以遍历大致同样数量的网页。

（2）蝴蝶结的左部（IN）。

这种类型的网页指向中心部分（SCC），称为“目录型网页”（hub page），即通常说的导航网页。Random-start BFS 遍历实验的起始点选择该部分网页，采用正向遍历的方法可以遍历占全部网页 3/4 的网页数；采用反向遍历的方法只能遍历很有限的一些网页，所占比例可以忽略不计。

（3）蝴蝶结的右部（OUT）。

这种类型的网页被中心部分指向，称为“权威性网页”（authority page）。这些网页被引用次数多，表示为大多数网页对其“认可度”高。Random-start BFS 遍历实验的起始点选择该部分网页，采用正向遍历的方法只能遍历有限的网页；采用反向遍历的方法可以遍历占全部网页 3/4 的网页。

（4） 蝴蝶结的须脚（Tendrils）。

这种类型的网页表现为从左部链出到其他网页，或者其他网页链入右部或从左部直接链入右部，以及少部分与中部、左部或右部都没有链接（非连通分量 DISC）。Random-start BFS 遍历实验的起始点选择这部分的网页，无论采用何种遍历方法都只能遍历有限的网页。

通过万维网的结构特征得出如下两个结论。

（1） 爬虫尽可能选择蝴蝶结的左部，或者中部的网页为起始访问结点集合（starting set of URLs）进行遍历，这样可以得到尽可能完整的遍历效果。如果选择右部或者须脚部分的网页为起始结点，则只能抓取很有限的网页。

（2） 网页分为目录型网页和权威性网页，目录型网页为普通网民服务，便于网民点击从而继续浏览更多的网页。该部分网页对于深入抓取权威性网页有重大意义；权威性网页是那些处于蝴蝶结中部或者右部的网页，这类网页的反向链接数很多，而正向链接数相对较少，通常认为这类网页比较重要。

万维网的直径

万维网直径（Diameter of the World Wide Web）又称为“Web 直径”，其定义为如果用 d 表示存在一条从网页 u 到网页 v 的路径，那么这些万维网上所有不同的连通网页对（pair of connected

page）所构成的最短路径的平均长度即 Web 直径。按照这样的定义和大规模网页的计算得到 Web 直径约为 17 左右。有兴趣的读者可以参考文献[Reka Albert et al. 1999]。万维网直径计算公式 d = 0.35 + 2.06 log（N）（其中 N 为网页的数量）。有研究表明中国万维网的直径为 16.26，即如果任何两个网页之间存在一条路径可达的话，平均点击不超过 17 次即可从一个网页到达另一个网页。

事实上，网页的平均出度为 25.7，即平均每一个网页含有 25.7 个指向其他网页的链接。这些链接引导网民继续在网上冲浪，直到对某个主题厌倦，开始下一个主题。下一章中还会讲到这个冲浪模型。

同样得到如下两个结论。

（1） 遍历的方法很大程度上影响了爬虫的效率，万维网的网页结构并没有我们想像的深，却出乎我们意料地宽，因此爬虫的遍历方式多采用宽度优先的遍历方式。当然这里还有一个网页重要性的原因，采用这种方式可以较好地抓取重要性高的网页。

（2） 万维网错综复杂，任选一个抓取路线不能保证总是最优。为了防止爬虫一路走到黑，充分考虑万维网的万维网直径后，采用“深度策略”控制抓取深度，从而完美地解决了这个问题。

万维网的规模及变化特征

在了解了万维网宏观静态的一面之后，再来看看其宏观动态变化的一面。

毫无疑问，万维网的规模是巨大的。早在 1998 年，各种研究就在试图估计万维网的规模。尽管方法各异得出的具体数值不尽相同，但都认为万维网的规模（有价值的网页数目）在十亿数量级上。根据 Lawrence 和 Giles 在 1999 年研究显示，世界万维网网页的数目每两年增加一倍，因此截至目前万维网的规模已达百亿数量级。

2000 年斯坦福大学的 Cho 和 Garcia-Molina 随机选用 50 万个网页做样本，发现 23%的网页是每天更新的，其中 40%域名后缀为.com 的网页是每天更新的，网页的半衰期（half-life）为 10 天。也就是说 50 万的网页在 10 天后只剩下 25 万，再过 10 天只剩下 12.5 万。此外，网页的变化可以归结为泊松过程（Poisson process）模型（在网页重访策略部分会详细提到这个模型）。可以说万维网的变化是日新月异，一日千里，如何与万维网的变化保持同步成为爬虫的又一个主要难点。

网页的特征

让我们从宏观来到微观，考察一下网页的特性。归纳起来，对于搜索引擎来说，网页具有的 3 大特征分别是挥发性（volatile）、半结构性（semi-structed）和隐蔽性（hidden）。网页诞生到网页

消亡（挥发性）总是不可避免的，HTML 语言描述的网页是一种半结构化的数据。除了静态网页以外，还有很多隐藏的动态网页，例如需要登录才能看到的某些动态网页。

接下来从有关爬虫的基本概念开始，由点到面，层层深入地理解爬虫是如何针对万维网的这些特点，采取相应的策略实现了抓得全、抓得快且代价低的基本目标。

第三节 有关爬虫的基本概念 03

爬虫

爬虫也称为“Wanderers”(漫步者)或者“Robots”(机器人),它首先是一组运行在计算机中的程序，在搜索引擎系统中负责抓取时新的且公共可访问的 Web 网页、图片和文档等资源。这种抓取的过程为通过下载一个网页，分析其中的链接。继而漫游到其他链接指向的网页，循环往复。

种子站点

种子站点是爬虫开始抓取的起点，通常为各大门户网站和官方网站的首页等。

URL

URL 是 "Uniform Resource Locator"（统一资源定位器）的缩写，它是用在万维网和其他万维网资源中的一种编址系统，例如 http://www. nju.edu.cn/index.html，其中包含访问方式（http 协议）、被访问的服务器（www.nju.edu.cn）和被访问的文件（index.html）。

Backlinks

一个网页的 Backlinks 是那些除网页自身之外指向自身链接的集合，Backlinks 的数目是衡量网页受"欢迎"程度的重要度量方式。

第四节 网页抓取原理 03

爬虫的工作原理包括抓取、策略和存储，抓取是爬虫的基本劳动过程；策略是爬虫的智慧中枢；存储是爬虫的劳动成果。我们按照逐个击破，由浅入深的方式逐步了解整个爬虫的工作原理。

telnet 和 wget

使用 Windows 操作系统的用户，执行如下步骤即可下载一个网页。

（1） 打开 Windows 命令行窗口。

（2） 输入 telnet www.nju.edu.cn 80（注意中间的空格）。

（3） 输入 GET /index.html（注意 GET 要全大写，其后要有空格）。

如果以上每一步都正确执行，则应该在桌面上看到南京大学网站首页的网页源代码。笔者得到的是如下的网页代码：

```
<script>location.href="./cps/site/NJU/nju/nju
.htm"</script>
```

使用 Linux 操作系统的用户，则只需要一步，即输入：vim http://www.nju.edu.cn/index.html，可以得到同样的结果。

如果要把该网页文件下载到本地硬盘，对于 Linux 操作系统的用户，只需要输入命令：

```
wget www.nju.edu.cn/index.html
```

之后使用 vi 可以打开该文件。Windows 操作系统的用户可以下载一个 wget 程序，使用同样方法下载网页。

由此看来，下载一个网页如此简单，如果要下载整个万维网，那么应当采用什么样的遍历规则呢？

从种子站点开始逐层抓取

基于万维网的蝴蝶结型结构，这种非线性的网页组织结构，就出现了抓取“顺序”的问题，即哪些先抓、那些后抓。这种解决抓取“顺序”的策略必须保证尽可能抓取所有网页（本章不区分抓取网页和下载网页的区别）。

一般来说，爬虫选择蝴蝶结左部的网页。即目录型网页作为

种子站点（抓取出发点），典型的如 sina.com 和 sohu.com 这样的门户网站的主页。每完成一次抓取网页之后提取其中的链接（提取的方法需要一些 HTML 语法分析，以及区分绝对路径和相对路径的技巧等），这些字符串形式的链接是指向其他网页的 URL，它们指引爬虫更加深入地抓取其它网页。一个网页常常包含多个链接，因此在提取网页的链接后，如何继续抓取其他网页，爬虫有如下两种选择处理抓取的“顺序”问题。

（1） 深度优先策略（Depth-First Traversal）。

深度优先的遍历策略类似中华文化中家族继承的策略，典型的如封建帝位的继承。通常为长子继承，如果长子过世，长孙的优先级大于次子的优先级。如长子过世且长子无子，那么次子继承，这种在继承上的优先关系也称作深度优先策略。如图 3-2 所示。

读书笔记

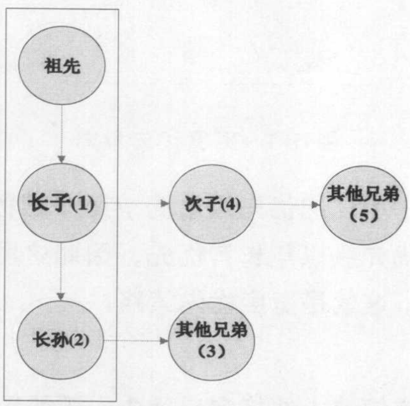


图 3-2　深度优先策略

在继承优先顺序上，长子>长孙>长孙的其他兄弟>次子>次子

的其他兄弟。这种首先选择某个分支，继而深入到不能深入的情况下才考虑其他分支的策略即为深度优先策略。

（2） 宽度优先策略（Breadth-First Traversal）。

宽度优先也称为“广度优先”，或“层次优先”，它是一种层次型距离不断增大的遍历方式，类似长幼有序的规则。在晚辈给长辈献茶时，总是先献长辈，然后次之，如图 3-3 所示。

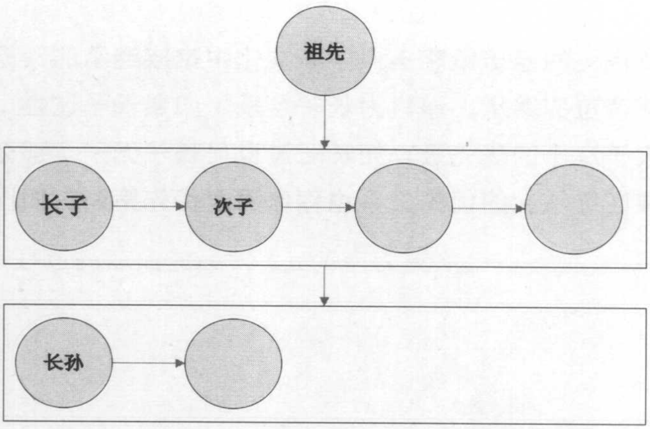


图 3-3　宽度优先策略

在图 3-3 中，祖先的优先级最高，第 2 层的优先级大于第 3 层，每层的内部优先级以年长者优先。因此这里次子的优先级大于长孙的优先权，这就是宽度优先策略。

在抓取的顺序策略上选择宽度优先出于如下 3 点原因。

首先，重要的网页往往离种子站点的距离较近，这符合直觉。我们通常在打开某些新闻网站时，进入眼帘的往往是最重要的新

读书笔记

闻。随着不断地冲浪（可以理解为深度不断加深），所看到的网页的重要性越来越低，甚至偶尔会出现无法访问的情况。

其次，万维网的深度没有我们想像的那么深，到达某一个网页的路径通常很多，总会存在一条很短的路径到达。有研究表明，中文万维网直径的长度只有 17。

最后，宽度优先规则有利于多爬虫合作抓取（这种合作策略在后面还会提到）。这是因为该规则开始抓取的网页通常都是站内网页，逐渐才会遇到站外链接，因此抓取的封闭性较强。

进行宽度优先遍历时，必须要有一个队列（queue）数据结构支持。这个队列理解为其工作负载队列，只要其中存在没有完成的抓取任务，就需要提取队头位置的网页继续抓取。直到完成全部抓取任务，工作负载队列为空为止。详细的抓取的过程如图 3-4 所示。

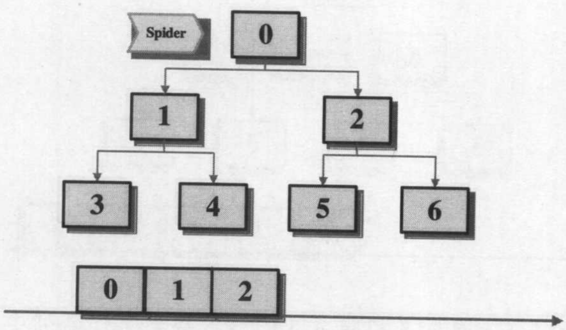


图 3-4　详细的抓取过程

（1）爬虫提取工作负载队列中的网页 0（这里假定网页 0 是种子站点，即抓取的起始结点），抓取后通过对网页 0 的链接分

析提取指向网页 1 和网页 2 的链接放到工作队列中。此时当前工作负载队列中增加了网页 1 和网页 2，如图 3-5 所示。

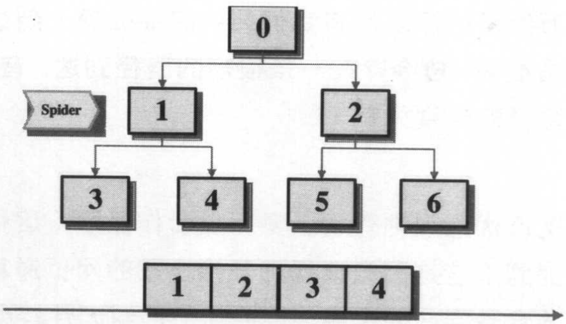


图 3-5　工作负载队列中增加了网页 1 和网页 2

（2） 工作负载队列中的网页 0 抓取完毕，继续抓取工作队列中第 1 个任务，即网页 1。并提取指向网页 3 和网页 4 的链接放到工作队列中，如图 3-6 所示。

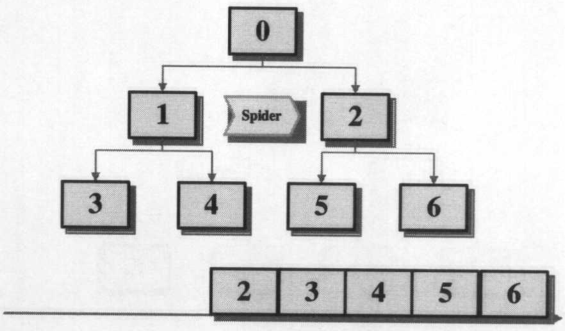


图 3-6　网页 3 和网页 4 的链接放到工作队列中

（3） 根据宽度优先的规则抓取工作队列中的网页 2（注意，这里可以理解为次子的优先级大于长孙，即网页 2 要在网页 3 之前抓取）,并且分析出指向网页 5 和网页 6 的链接放入工作队列中，

如图 3-7 所示。

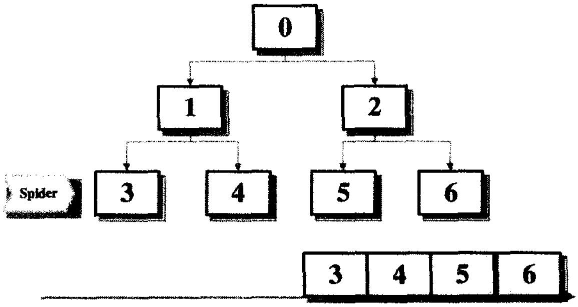


图 3-7　网页 5 和网页 6 的链接放入工作队列中

（4）继续抓取工作队列中的网页 3，直到结束，如图 3-8 所示。

如果将余下的步骤略去，这种按宽度优先顺序的抓取序列就展示在我们的眼前了，即网页 0~6。

通过这个例子回顾一下采用宽度优先遍历的策略抓取网页带来的好处。

如果将网页 0 理解为门户首页，那么距离门户首页越近的网页越重要。把首页理解成一个窗口，那么打开这个窗口，距离越近的网页被浏览的机会越大，因此也就越重要。上面这个例子中网页 1 比网页 3 和网页 5 更加重要，按照距离递增的宽度优先抓取顺序恰好符合了重要网页优先抓取的要求。如果按照深度优先规则抓取，不仅破坏了重要优先的原则，而且破坏了抓取的封闭性。从而不利于多爬虫的合作抓取，在合作抓取策略中还将提到抓取封闭性问题。

然而，以上的例子是不完备的。为了简化描述，这里使用树型结构。树的一个基本特性是任何两点之间只有惟一的路径，即不会出现环路。而万维网中存在大量环路，实际上网页 6 可能存在指向网页 0 的链接。如果没有任何判断，则爬虫永远抓取不完，因为存在了 0、2 和 6 这样一个死循环。死循环会带来下面两个不利后果。

- 不该抓的反复抓。如果一个网页变化不大，则只要抓取一次即可，重复抓取相同网页就会占用大量 CPU 和带宽资源。
- 该抓的没有机会抓。由于大量的资源被用在做无用功，所以必然使得某些网页被歧视，从而不能被及时抓取。

为了解决死循环的问题，通常需要一个称为“不重复抓取策略”和一个称为“深度策略”的方法来解决。

不重复抓取策略

不重复的关键在于记住历史，只有记住过去才可能不重复。那些不需要记忆历史的程序设计往往都比较简单，从计算机的角度上讲即有限状态机模型。每个状态都是对历史的积累，而没有记忆历史，这种只需要有限的存储即可完成的工作通常都比较简单。而那些需要记住历史的程序相对复杂，例如迷宫求解程序，则需要一个栈（stack）结构记住走过的地方。从而在失败时可以回溯，继续寻找出路。而通过动态规划寻找问题最优解等程序，也都需要一个记住历史的功能才能保证不重复计算。

爬虫记录历史的方式是哈希表（也称为“杂凑表”），每一条记录是否被抓取的信息存放在哈希表的某一个槽位上。如果某网页在过去的某个时刻已经被抓取，则将其对应的槽位的值置 1；反之置 0。而具体映射到那一个槽位，则由哈希函数决定。在介绍哈希表前，我们首先简单了解一下 MD5 签名函数。

1992 年 8 月，Ronald L. Rivest 在向 IEFT 提交的一份重要文件中描述了这种 MD5 签名算法的原理。由于这种算法的公开性和安全性，所以在 90 年代被广泛使用。MD5 签名是一个哈希函数，可以将任意长度的数据流转换为一个固定长度的数字（通常为 4 个整型，即 128 位）。这个数字称为“数据流的签名”或者“指纹”（Digital Finger Print），并且数据流中的任意一个微小的变化都会导致签名值发生变化。

将 URL 字符串数字化是通过某种计算将任何一个 URL 字符串惟一地计算成一个整数，如图 3-8 所示。

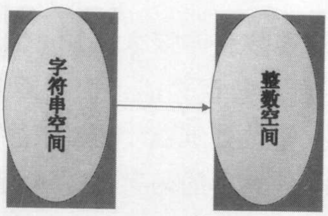


图 3-8　字符串数字化

在一个 URL 哈希函数映射下，任意一个字符串都惟一地对应一个整数。一个 64 位整数可以表达 1 800 万 TB（1TB = 1 000 GB），而字符串空间的大小远远大于 64 位整数所表达的整数空间大小，因此碰撞是不可避免的。碰撞是指那些字符串不同，而计算出相同的签名值的情况。然而如果杂凑函数设计得足够好，则相互碰

撞的机会可以小到忽略不计。

更加理论化的表示如下。

令 U=128 位整数集合，S =字符串集合：

$$\forall URLi \in S, URLj \in S$$
$$Ti = MD5(URLi)$$
$$Tj = MD5(URLj)$$

其中 $Ti \in U, Tj \in U$ ，$URLi \neq \mathrm{URL}j$ ，则 $P(Ti = Tj) < \varepsilon$ 。

不难看出，对于两个不相同的 URL，即 URLi 和 URLj 通过哈希函数分别计算出各自的签名值 Ti 及 Tj，这两个签名值相等（碰撞）的概率小于一个足够小的小整数 ε。正是由于 MD5 函数符合这种特性，因此被广泛地用做哈希函数。

标准 MD5 签名的整数空间很大，128 位整数能表达 2 的 128 次方个不同的数，这是十分巨大的，而实际分配的哈希表空间确十分有限，一个普通 32 位处理器，理论上最多可以分配 2 的 32 次方大小的内存，即 4G 大小的内存（Linux 系统中操作系统内核还需占用 1G 内存，实际上可供搜索引擎使用的极限内存只有 3G）。

因此，在实际处理中将签名值进行模运算映射到实际的哈希表中（可以理解为哈希表存放在内存中）。因此实际的哈希函数是 MD5（URL）%n（%表示取模运算），这样使得一个 URL 被映射到大小为 n 的哈希表的某个槽位上。

哈希表是简单的顺序表，即数组。从实际应用的角度看，这个数组要足够大。而且尽可能地全部放在内存中，以保证每一个 URL 的签名都可以通过查找哈希表来确定是否曾经抓取过，其示例如图 3-9 所示。

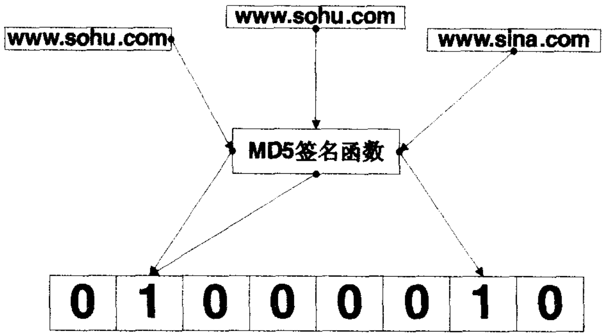


图 3-9　哈希函数示例

假定抓取顺序为 www.sohu.com、www.sina.com 和 www.sohu.com，首先我们在抓取 www.sohu.com 时为该 URL 签名。假定得到的结果为 2，我们存放在数组的第 2 个槽上。这个过程只要将第 2 个槽位置 1，表示已访问过。继续抓取 www.sina.com，以同样方法，假定将其存放在数组的第 7 个槽位上，之后重复抓取到了 www.sohu.com。由于 MD5 签名函数特性，任何一个自变量惟一对应一个应变量。因此计算 www.sohu.com 的签名，同样得到结果为 2。当我们查询第 2 个槽位时，发现该槽位已经置 1。说明已经访问过，因此不再重复访问。

有调查表明，中文万维网页总数超过 100 亿。如果要存放 100 亿网页是否被抓取过这样一个历史信息，那么需要的哈希表将会非常大。如果哈希表每一个单元为一个字节（8 位），则至少需要 10 GB 的容量。而且为了保证碰撞的概率极低，实际需要的容量

远大于 10 GB。然而 32 位操作系统的最大寻址空间为 4 GB，即理论上最大的物理内存为 4 GB，因此在内存中不可能分配一个如此巨大的哈希表。

为了能够将整个哈希表装入内存，通常采用 Bitmap 的数据结构压缩内存用量，Bitmap 需要借助按位与（&）和按位或（|），以及左移（<<），右移（>>）这 4 种基本位运算。通过一个例子来说明，如图 3-10 所示。

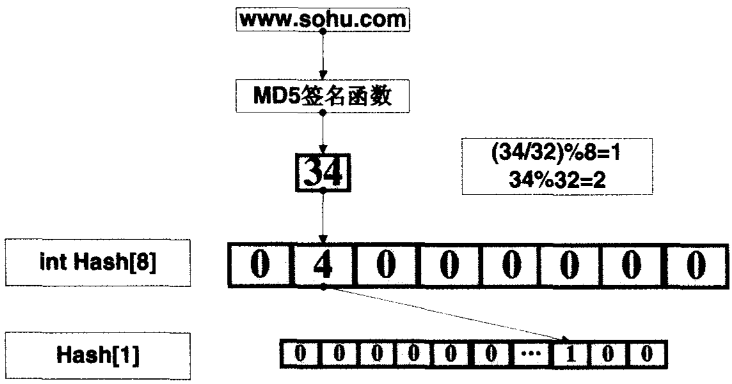


图 3-10　Bitmap 结构的哈希表

Bitmap 结构使用整型作为基本单元，一个整型为 32 位，一个 int Hash[8]的数组仅通过 8 个单元来记录历史。而如果将比特作为最小单元，则能扩展到 256 个单元。

定义这样一个哈希表，即 int Hash[8]。将“www.sohu.com”作为字符串执行一次 MD5 签名，假定得到的签名值为 34。用 34 除 32，得到商数为 1，继续用 1 对 8 取模运算得到 1，表示槽位在 Hash[1]中。用 34 除 32，得到余数为 2，表示 34 映射到 Hash[1]这个 32 位整数的第 3 个比特位上（从低位算起，余数为 0 则映射

到第 1 个比特位），将这个比特位置 1，则 Hash[1]为 4，这是因为 Hash[1]的第 3 个比特位被置 1。

不妨将 34 这个数按比特位的方式展开，如图 3-11 所示。通过这种形式进一步理解 Bitmap 如何扩展存储空间的方法。

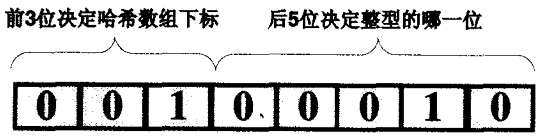


图 3-11 将 34 位按比特位表达

34 的二进制的前 3 位 001 决定了它存放在数组的第 2 个位置上（数组的起始下标为 0），即存放在 Hash[1]上，前面提到的除以 32 后，对 8 取模的目的就是取出 34 这个数的前 3 位。

具体存放在 Hash[1]的从低位算起的第几个比特位则由后 5 位决定。后 5 位为 00010，表示存放在 Hash[1]这个整数的第 3 个比特位上。前面提到的 34 除以 32 后得到余数即为这里的后 5 位。

通过前面的计算，Hash[1]的实际值为 4，表示其第 3 个比特位被置 1。图 3-11 中，在 Hash[1]中用 4 这个数值完整地表示了 34 这个签名值。

通常使用移位运算来快速计算除以或者乘以一个 2 的 n 次幂；将某个比特位置为 1，采用按位或运算；查询某个比特位是否为 1，

采用按位与运算。

对前面计算的过程用 C 代码的方式表示如下：

（1） int MD5 = 34;

（2） int index_int = MD5&31; // 相当于 34%32

（3） int index_Hash =（ MD5>>5 ）&7;//相当于（ 34/32 ）%8，32 是 2 的 5 次幂

（4） if（ Hash[index_Hash]&（ 1<<index_int ）） //表示探测该槽位是否已经被置位，注意这里的按位与运算，查询某个比特位是否为 1 的技巧，1<<index_int 表示将 1 右移 index_int 位。

（5） else{ Hash[index_Hash]= Hash[index_ Hash] |（ 1<<index_int ）; //表示该槽位没有被置位，用或运算置位。

上述过程读者可以用纸笔计算一下，从而了解这种解决问题的技巧。通过 Bitmap 结构，利用计算机速度极快的按位与、按位或和移位运算能够高效地实现记住历史，不重复抓取的策略，同时将原来的哈希表压缩了 1/32。

除此之外，Bloom filter 算法还可以更加经济地利用好哈希表中的比特位，使得更加经济地灵活地使用内存。简单地说，对于一个字符串独立地使用 m 哈希函数计算，然后将其值映射到哈希表的 m 槽上。如果这 m 个槽都置 1，说明该字符串已经在历史上

出现过；否则说明该字符串是第 1 次出现，将 m 个槽都置 1。通过图 3-12 来进一步理解。

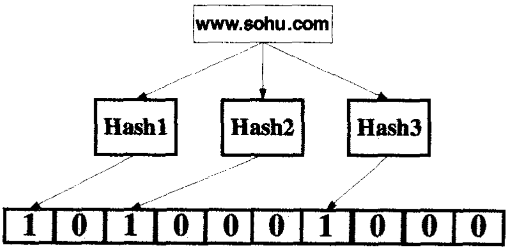


图 3-12　Bloom filter 原理

在抓取到 www.sohu.com 时，执行 3 次独立的不同的哈希函数（如图 3-12 中的 Hash1、Hash2 和 Hash3）运算。映射到哈希表的 3 个槽上，假定是 1、3 和 7。接下来如果抓取到 www.sina.com。经过同样的 3 次哈希函数运算，假定计算的结果是 1、3 和 6。很明显存在一个槽 6 是之前没有置位的，因此 www.sina.com 没有抓取过。成功抓取 www.sina.com 后，将槽位 6 置 1。Bloom 算法一方面提高了哈希数组的利用效率，并被证明适宜于应用在大规模集合查找计算中。同时在网络应用中也被大量采用，有兴趣的读者可以进一步阅读文献 [Bloom，B.H. 1970]深入理解该算法。

在抓取提速部分讲到通过多爬虫的合作抓取，这样可以进一步降低每个爬虫的用于记录历史抓取情况的哈希表大小。即如果有 n 个爬虫，则可将哈希表继续压缩到原有大小的 n 分之一。如果 n 个爬虫分别运行在不同的机器上，那么每个机器被哈希表占用的内存用量将非常少，通常保持抓取历史记录所需要的内存在百兆字节左右是恰当的。

通过不重复抓取的方法初步解决了死循环的问题，即抓过的

不再抓，循环的条件被破坏。然而实际操作中还有这样一个隐含的问题，即如前所述的万维网直径。如果任意两个网页存在链接，则链接它们的最短路径的期望值为 17。这样，爬虫无论用何种遍历方法都不能保证一定会按照最佳路径抓取每一个网页，因为任何一个网页都可能从多个种子站点开始宽度优先的被遍历到。

我们来看一个例子。

假如从种子站点 A、种子站点 B 及种子站点 C 开始均存在一条到达网页 P 的路径，路径长度分别为 3、27 和 143。很明显，需要浪费两次检查网页 P 是否被抓取的开销。由于从种子站点 A 开始很快抓到了网页 P，而种子站点 B 和 C 则经过漫长的路径达到 P 时共计需要两次检查网页 P 是否被抓取，这显然是不够经济的。

为了防止爬虫无限制的宽度优先抓取，必须在某个深度上进行限制。到达这个深度后就应该停止抓取，这个深度的取值就是万维网直径长度。当在最大深度上停止时，那些深度过大的未抓网页，总是期望可以从其他种子站点更加经济地到达。例如，种子站点 B 和 C 在抓取到深度 17 时即停止抓取，把抓取网页 P 的机会留给从种子站点 A 出发的进行抓取工作的爬虫。不难看出，限制抓取深度也破坏了死循环的条件，即使出现循环也会在有限次后停止。此外，深度策略和宽度优先遍历策略的组合可以有效地保证抓取过程中的封闭性。即在抓取过程（遍历路径）中总是在抓取相同域名下的网页，而很少出现其他域名下的网页。

爬虫总在不停地抓取网页，其工作负荷能力总是有限的。同时下载带宽和时间也是有限的，因此必须能够优先下载重要性高

的网页。下面的网页抓取顺序策略保证了即使不能下载全部网页也要保证能下载全部重要性高的网页。

网页抓取优先策略

网页抓取优先策略也称为“页面选择问题”(Page Selection)，通常是尽可能地首先抓取重要性的网页，这样保证在有限的资源内尽可能地照顾到那些重要性高的网页。那么哪些网页才是重要性高的呢？如何量化重要性呢？

重要性度量由链接欢迎度、链接重要度和平均链接深度这 3 个方面决定[Arvind Arasu et al. 2001]。

定义链接欢迎度为 IB(P)，它主要由反向链接(Backlinks)的数目和质量决定。首先考察数目，直观地讲，一个网页有越多的链接指向它(反向链接数多)，那么表示其他网页对其的认可。同时这个网页被网民访问的机会就大，推测出其重要性也就越高；其次考察质量，如果被越多重要性高的网页指向，那么其重要性也就越高。如果不考虑质量，就会出现局部最优，而不是全局最优的问题。最典型的就是作弊网页，人为地在一些网页中设置了大量反向链接指向其自身的网页，以提高该网页的重要性。如果不考虑链接质量，就会被这些作弊者所利用。

定义链接重要度为 IL(P)，它是一个关于 URL 字符串的函数，仅仅考察字符串本身。链接重要度主要通过一些模式，比如认为包含“.com”或者“home”的 URL 重要度高，以及具有较少斜杠(slash)的 URL 重要度高等。

定义平均链接深度为 $ID(P)$，此为笔者所创。$ID(P)$表示在一个种子站点集合中，每个种子站点如果存在一条链路（宽度优先遍历规则）到达该网页，那么平均链接深度就是这个网页的又一个重要性指标。因为距离种子站点越近，说明被访问的机会越多。因此重要性越高，可以认为种子站点是那些重要性最高的网页；离种子站点越远，重要性越低。事实上，按照宽度优先的遍历规则即可满足这种重要性高的网页被优先抓取的需要。

最后，定义网页重要性的度量为 $I(P)$，它由以上两个量化值线性决定，即：

$$I(P) = \alpha * IB(p) + \beta * IL(p)$$

平均链接深度由宽度优先的遍历规则保证，因此不作为重要性评价的指标。在抓取能力有限的情况下，如果能够把重要性高的网页尽可能地抓完，是合理科学的，最终被用户查询到的网页也往往是那些重要性高的网页。

尽管这样看来已经足够完美，事实上，还是忽视了一个重要的要素——时间。时间导致万维网动态变化的一面。如何抓取那些新增的网页呢？如何重访（revisit）那些被修改了的网页呢？如何发现那些被删除了的网页呢？为了保持和万维网网页的同步变化，就必须有网页重访策略。通过该策略可以识别增加、修改及删除网页这 3 种网页变化的情况。

网页重访策略★

爬虫日积月累地下载各种各样的网页，然而过去的网页可能变化了。因此爬虫不得不周期性的刷新（refresh），重访那些已经下载的网页。通过重访以使得这些网页能够与万维网的变化与时俱进（up-to-date）。

文献[CHO, J. AND GARCIA-MOLINA, H. 2000a]表明网页变化的可以归结为泊松过程（Poisson process）模型。

为了描述泊松过程模型，使用 $X(t)$ 表示在一个（$0,t$）的时间段内网页发生变化的数目，一个参数为 λ 的泊松分布满足以下性质。

对于 $s \geq 0$，$t \geq 0$，随机变量 $X(s+t)-X(s)$ 符合泊松分布，即：

$$\Pr\{X(s+t)-X(s)=k\}=\frac{(\lambda t)^k}{k!}e^{-\lambda t}$$

其中 $k=1,2,3\cdots$

随机变量 $X(s+t)-X(s)$ 的期望值为 λt：

$$E[X(s+t)-X(s)]=\lambda t$$

可以通过简单的方法验证，假定时间周期（时间间隔）为 *1*（t=1），则有：

$$E[X(t+1)-X(t)]=\sum_{k=0}^{\infty}k\Pr\{X(t+1)-X(t)\}=\sum_{k=1}^{\infty}k\frac{\lambda^k e^{-\lambda}}{k!}=\lambda$$

其中：$\sum_{k=1}^{\infty}\frac{\lambda^{k-1}}{(k-1)!}=e^{\lambda}$

采用 $E[X(s+t)-X(s)]=\lambda t$ 的计算公式同样也得到了

$$E[X(t+1)-X(t)]=\lambda t=\lambda$$

Cho 和 Garcia-Molina 通过对 50 万个随机网页的跟踪分析，得出大部分网页的更新属于泊松分布的重要结论，如图 3-13 所示。

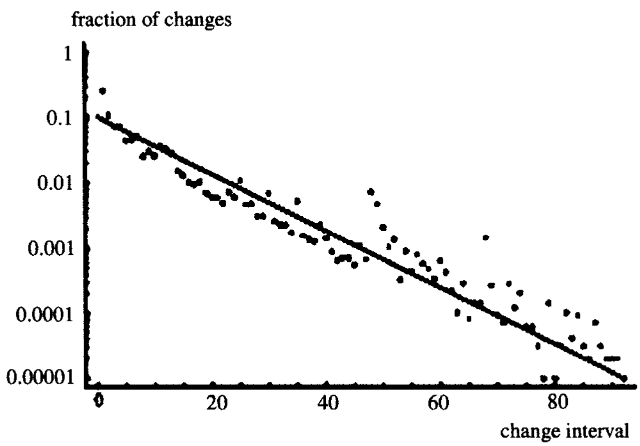


图 3-13　网页更新的泊松分布[CHO et al.2000a]

横坐标表示网页更新的事件间隔，纵坐标是在时间间隔下发生变化的网页所占比例（该比例取对数）。

在泊松模型下，若 t 是泊松过程中下一个事件发生的间隔时间，则下一个事件发生的时间间隔的概率分布符合指数分布。在这里表示网页变化的时间间隔符合指数分布，即：

$$\varphi(t)=\begin{cases}\mu e^{-\mu t}, t>0\\0, t\le 0\end{cases}$$

概率分布可以这样理解，在泊松过程中，下一个事件发生的时间间隔不超过 t 的概率为 $\int_0^t \varphi(t)$。这里注意概率和概率分布的区别。

对 $\varphi(t)$取自然对数，得到 $\ln(\varphi(t))=-\mu t\ln(\mu)$，因此 $\ln(\varphi(t))$ 是一个关于 t 的线性函数，如图 3-13 所示。直观上可以看到，网页变化的可能性随着变化周期（时间间隔）变长越来越小，文献[CHO, J. AND GARCIA-MOLINA, H. 2000c]较完整地论述了关于变化间隔的计算方法及其他有趣的话题。

公式中的 t 表示网页发生变化的时间间隔，求 t 的期望寿命：

$$E(t)=\int_0^\infty t*\mu*e^{-\mu t}dt=\frac{1}{\mu}$$

这样如果把平均的时间间隔看做是网页寿命，则μ的倒数就是网页的期望寿命。

在泊松模型的理论基础上，结合人们的直观感知，目前网页重访策略大致可以归为以下两类。

（1）统一的重访策略：爬虫以同样的频率重访已经抓取的全部网页，以获得统一的更新机会，所有的网页不加区别地按照同样的频率被爬虫重访。

（2）个体的重访策略：不同网页的改变频率不同，爬虫根据其更新频率来决定重访该个体页面的频率。即对每一个页面都量身定做一个爬虫重访频率，并且网页的变化频率与重访频率的比率对任何个体网页来说都是相等的。即对于任意网页 P_i，其固有的更新频率 λ_i（次数/单位时间）。爬虫对其重访的频率为 f_i，则 $\frac{\lambda_i}{f_i}$ 为一个常数。即对于网页 P_i 和 P_j，有 $\frac{\lambda_i}{f_i}=\frac{\lambda_j}{f_j}$（$i \neq j$）。那些更新频率高的网页，重访频率就高；更新频率低的网页，重访频率就低。

两种重访策略各有利弊，深入的研究表明对于那些更新频繁的网页采用更加频繁的重访策略反而有害[CHO, J. AND GARCIA-MOLINA, H. 2000b]。因此策略（2）还需要增加一些其他子策略，例如限定最小重抓频率等。策略（1）的效果在数学上可以证明总是不低于个体重访策略[CHO, J. AND GARCIA-MOLINA, H. 2000b]，并且具有计算量小的优势。但是策略（1）不具有优化的空间，难以满足质量不断改善的要求。

此外，不同类型的网站更新频率也不同。通过对不同类型的网页寿命的半衰期实验证明了这一点，网页更新的泊松分布如图3-14 所示[CHO et al.2000c]。

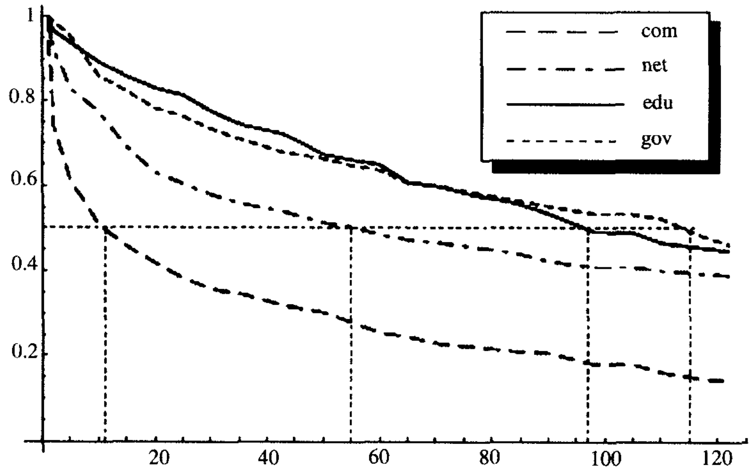


图 3-14 网页更新的泊松分布

通过该图，得到.com 类型的网页变化最剧烈，.gov 类型的网页变化最少，因此给予不同类型的网页不同的重访频率是科学的。

回顾网页更新策略既能保证抓取历史的网页，也能够抓取随时出现的新网页，几个重要的结论如下。

（1） 网页更新过程符合泊松过程。

（2） 网页更新时间间隔符合指数分布。

（3） 对于不同类型的网页采用不同的更新策略。

粗放的工作做完了，然而还远远没有达到完美。爬虫的工作

模式和方法存在一些问题，甚至可以说是社会问题 [BRIN, S. AND PAGE, L. 1998]。爬虫的工作方式某些方面上看是脆弱的，不可控的。因为在其中包含了数以千计的万维网上的 Web 主机，以及大量的 DNS 服务器，这些都是爬虫所不能控制的。只有协调好这些外部资源，才能达到理想的抓取效果。

有些网站不希望爬虫抓取敏感信息；有些网站希望爬虫避免在白天对其网页进行抓取，从而不影响白天正常的对外公众服务；DNS 服务提供商也不希望大量的域名解析工作量被搜索爬虫的域名请求所占用。

因为爬虫的工作可能带来的社会性的问题，为此要求爬虫一方面像绅士一样工作，了解网站管理员的需要。例如有些 Web 主机不希望爬虫抓取某些敏感目录，或者网页，这样网站管理员必须有办法告诉爬虫哪些是不希望被抓取的网页或目录。爬虫也需要礼貌地按照对方的要求执行，这就是著名的 Robots 协议；另一方面，爬虫尽可能自身维护一台 DNS 解析的服务器，而减少访问公开 DNS 域名解析服务器的机会；最后，抓取的工作尽可能放在夜间，从而减少 Web 主机在白天的访问压力。

不同 Web 主机的带宽不同，能够接受的抓取强度也不同。爬虫需要尽可能平衡地抓取 Web 主机上的网页，即将集中抓取平衡化；反之，如果集中抓取某一个站点，很可能会把某个 Web 主机抓瘫痪，而影响 Web 站点的正常公众服务。

遵守 Robots 协议是爬虫礼貌工作的第 1 步。

Robots 协议

Robots 协议是 Web 站点和搜索引擎爬虫交互的一种方式，即将一个 robots.txt 的文件放在网站的根目录上，例如 http://www.w3.org/robots.txt。

该文件的内容如下：

```
# robots.txt for http://www.w3.org/
# $Id: robots.txt,v 1.45 2006/06/05 01:11:19 ted Exp $
# For use by search.w3.org
User-agent: W3C-gsa
Disallow: /Out-Of-Date
User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT; MS Search 4.0 Robot)
Disallow: /
# W3C Link checker
User-agent: W3C-checklink
Disallow:
# exclude some access-controlled areas
User-agent: *
Disallow: /2004/ontaria/basic
Disallow: /Team
Disallow: /Project
Disallow: /Systems
```

```
Disallow: /Web
Disallow: /History
Disallow: /Out-Of-Date
Disallow: /2002/02/mid
Disallow: /mid/
Disallow: /People/all/
Disallow: /RDF/Validator/ARPServlet
Disallow: /2003/03/Translations/byLanguage
Disallow: /2003/03/Translations/byTechnology
Disallow: /2005/11/Translations/Query
Disallow: /2003/glossary/subglossary/
#Disallow: /2001/07/pubrules-checker
#shouldnt get transparent proxies but will ml
links of things like pubrules
Disallow: /2000/06/webdata/xslt
Disallow: /2000/09/webdata/xslt
Disallow: /2005/08/online_xslt/xslt
Disallow: /Bugs/
Disallow: /Search/Mail/Public/
Disallow: /2006/02/chartergen
```

在上述文件中，Robots 协议通过 User-agent 和 Disallow 告知搜索引擎非公开目录和非公开网页，说明如下。

（1） User-agent:*：表示对一切搜索引擎爬虫有效，如果特别针对某个爬虫，则可以写明。

（2）Disallow:/ 2004/ontaria/basic：表示禁止抓取这个目录。

通过遵守 Robots 协议，表示出爬虫尊重和执行 Web 站点的要求。因此爬虫需要有一个分析 Robots 协议的模块，并严格按照 Robots 协议的规定只抓取 Web 主机允许访问的目录和网页。

其他应该注意的礼貌性问题

除了遵守 Robots 协议以外，通常搜索爬虫需要自身维护一个 DNS 域名解析的服务模块，以便减少对公开域名解析服务器的频繁请求，本书不展开讨论这部分内容。

此外，应尽可能合理地规划抓取强度。在白天尽可能地减弱抓取强度，例如每 10 秒抓取一次；在夜间 Web 主机访问量低，搜索爬虫可以适当增加抓取强度，例如每秒抓取一次。由于时差的原因，东半球的白天恰好是西半球的夜晚，因此白天可以加强抓取美国及欧洲站点的强度，夜间增加对本国站点的抓取强度。

即便如此，爬虫总不可避免会给其他万维网上的 Web 主机带来麻烦。因此站点抓取监控程序也是必不可少的，该程序记录每个站点的抓取流量，避免在偶然情况下因抓取强度太大而导致的各种问题。

这样，最后的问题就只剩下规模问题。如此巨大规模的万维网，每天新增的大量网页需要及时地被抓取到。对于爬虫来说，一方面对历史的网页需要重抓；另一方面要及时抓到新增的网页，

在如此沉重的工作负荷下，必须提高抓取速度才能满足这种需要。

抓取提速策略（合作抓取策略）

提速基本可以采用下面几种方法。

（1） 提高抓取单个网页的速度。

（2） 尽可能减少不必要的抓取任务。

（3） 增加同时工作的爬虫数量。

事实证明，受到万维网发展水平的限制，第（1）种方法基本不可行，单个网页的抓取速度受到下载带宽的限制，在现有技术条件下很难任意提高；第（2）种方法难度很大，由于需要和万维网的变化保持紧密同步，所以冗余的抓取总是不可避免的。减少不必要的抓取会导致网页重访不及时，这样就不能快速同步目标网页的变化；第（3）种方法通过增加爬虫数量提高总体抓取速度是可行的。

在多个爬虫抓取的情况下，如何将工作量分解成为主要的问题。即要解决一个网页交给哪一个爬虫抓取？如果分工不明，很可能多个爬虫抓取了相同的网页，从而引入额外的开销。通常采用以下两种方法来进行抓取任务的分解。

（1） 通过 Web 主机的 IP 地址来分解，使某个爬虫仅抓取

某个地址段的网页。

（2） 通过网页的域名来分解，使某个爬虫仅抓取某个域名段的网页。

如何选择这两种方案呢？

万维网在网络基础设施中按照 IP 地址来确定主机位置，IP 地址为点分十进制数，难于记忆。由此采用了域名对 IP 地址进行一次映射，由于域名对人友好，于是出现了这样一些问题。即存在多个域名对应同样的 IP 的情况，对于中小网站来说，通常采用这种方法提供不同的 Web 服务。这主要出于经济的考虑，因为可以只配置一台服务器；而对于大型网站，如新浪和搜狐这些门户网站通常采用负载均衡的 IP 组技术，同样的域名对应于多个 IP 地址。一方面提高系统健壮性，一方面做到了负载均衡。

鉴于多域名对应相同 IP 和同域名对应多 IP 的情况，通常的做法是按照域名分解任务。即只要保证不重复抓取大型网站的网页，小型网站即便重复抓取也可以接受的策略分配任务。这种分配方法将不同的域名分配给不同的爬虫抓取，某一个爬虫只抓取“指定”的一个域名集合下的网页。例如 sina.com.cn 被“指定”交给 spider1 抓取，zoujingsousuoyinq.cn 被“指定”交给 spider2 抓取，zoujingsousuoyinqbook.cn 被“指定”交给 spider3 抓取等。

这两种方案的主要区别可以通过下面这两个例子进一步理解。

假定 zoujingsousuoyinq.cn 和 zoujingsousuo yinqbook.cn 是两个域名不同，但 IP 相同的网站，假定为 10.10.67.208。有这样两个网页，即 http:// www.zoujingsousuoyinq.cn /index.html 和 http:// www.zoujingsousuoyinqbook.cn/index. html。在经过域名解析后，实际上它们同为 http://10.10.67. 208/index.html。采用域名分解的方案，spider2 和 spider3 就会重复抓到这个网页。但是由于这两个站点信息较小，所以可以容忍因抓重而带来的损失。

反之，我们如果通过 IP 分配抓取任务的方案，例如 sina.com.cn(71.5.7.138)被“指定”交给 spider1 抓取 sina.com.cn (71.5.6.136)被“指定”交给 spider2 抓取，zoujingsousuoyinq.cn (10.10.67. 208) 被 “指定” 交给 spider3 抓取，zoujingsousuo yinqbook.cn (10.10.67.208) 被 “指定” 交给 spider3 抓取。

这种分配方案，对于不同域名指向相同 IP 的情况，例如 zoujingsousuoyinq.cn 和 zoujingsousuoyinqbook.cn 的抓取工作都由 spider3 完成，没有重复抓取问题。但是 sina.com.cn 对应多个 IP，所以被分配由 spider1 和 spider2 分别抓取，这样 spider1 和 spider2 抓取工作就相互重复。很显然，新浪属于大型网站，这种因为抓重带来的损失往往是巨大的。

通过比较，为了照顾大站按照域名分解的策略更加合理。在下载系统中，按照域名分解抓取任务（网页集合）的工作由一个称为“调度员”的模块来处理，通过域名分解将不同的网页调度给不同的爬虫进行抓取。因此，下载系统主要由爬虫和调度员构成。

形式化的调度分配方式如下：

首先假定有 n 个爬虫可以并行工作，并且定义一个可以提取 URL 域名的函数 domain，例如：

```
    URL = http://finance.sina.com.cn/g/20070317
/17163416577.shtml
    domain (URL) = finance.sina.com
```

说明如下。

（1）对于任意的 URL，利用 domain 函数提取 URL 的域名。

（2）用 MD5 签名函数签名域名，MD5 (domain (URL))。

（3）将 MD5 签名值对 n 取模运算，int spider_no = MD5 (domain (URL)) %n。

（4）该 URL 分配给编号为 spider_no 的爬虫进行抓取。

由于模运算可以实现将一个全集分成多个等价类，所以等价类的并集等于全集，且一个等价类中的元素必然不属于另一个等价类中。

等价关系形式化表述如下。

令 U 为全集，通过某种等价关系可以将集合 U 分别映射到集合 S_1，S_2，…，S_n 上，并且满足如下两个条件。

（1） $S_1 \cup S_2 \cup \ldots \cup S_n = U$。

（2） $\forall a \in S_i$，$\forall b \in S_j$，$S_i != S_j$，则有 $a!=b$。

通常 n 取 2 的整数幂，例如 4，8，16，32…取模时可以利用按位与（&）的方法快速求得，int spider_no = MD5（domain（URL））&（n-1）。通常对 2 的整数幂 n 取模，都可以用&（n-1）的方法（其中 n 必须是 2 的整数幂才可以进行这样的计算）来快速计算。

这种策略的好处在于每个爬虫的任务量尽可能地均匀分配，同一个域名必然只由一个爬虫抓取，所有爬虫的工作量组合就是全部的抓取任务。在介绍了爬虫和调度员之后，已经能够完整地理解搜索引擎下载系统的体系结构，如图 3-15 所示。

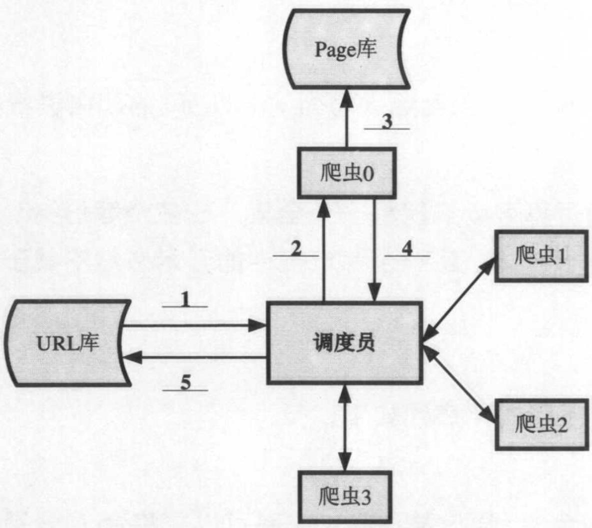


图 3-15　搜索引擎下载系统的体系结构

这里我们引入了 URL 库和 Page 库这两个概念，URL 库存放全部历史上曾经抓取过的 URL 和新增的 URL，Page 库存放爬虫实际抓取下来的原始网页。这样完整的合作抓取过程如下，注意下面的标号对应于上图中的数字。

（1）调度员通过更新规则向 URL 请求一个 URL 抓取任务。

（2）调度员计算出该 URL，然后分配给编号为 0 的爬虫抓取。

（3）爬虫 0 实际抓取的网页存放在 Page 库中。

（4）爬虫 0 在抓取的网页中提取其他链接后反馈给调度员。

（5）调度员判断网页类型，并设定初始更新时间等后存放在 URL 库中。继续转（1），周而复始。

在实际应用中，多采用多爬虫多调度员的体系结构，如图 3-16 所示。

值得注意的是，抓取的封闭性越强，对外的通信开销越小。如上图所示，假如爬虫 1 从 sina.com 这个种子站点开始抓取。由于总是抓取 sina.com 的网页，而这些网页总是应该归属爬虫 1 抓取，因此不需要和其他爬虫通信（不需要经过总调度）；反过来，如果抓取的封闭性差，表示可能抓到各种各样域名下的网页，并且可能需要交给其他爬虫抓取。总调度不得不把这些信息相互转发，这样就会增加额外的通信代价。因此提高抓取的封闭性可以

减少这种合作抓取带来的通信开销，前面提到过的宽度优先的遍历方法及深度策略能够有效的保证这种抓取的封闭性。

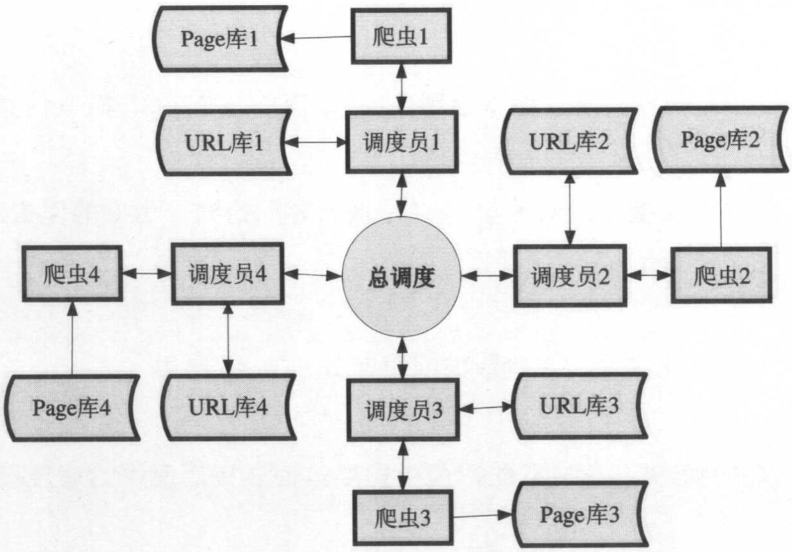


图 3-16 多爬虫多调度员的体系结构

最后，简单了解一下大规模网页存储的有关知识。

第五节 网页库 03

读书笔记

很显然，爬虫抓取的网页必须及时保存在硬盘上，因此网页库的首要挑战来自于能够快速存储大规模网页；其次这些网页必须能够被其他模块快速地读取，因此读写问题是围绕网页库的主要难点。

1. 可伸缩性（Scalability）

网页库的存储必须具有可伸缩性，可以将大规模网页平滑地分布在一组计算机的硬盘中。

2. 双访问模式（Dual access modes）

网页库必须能够有效地支持两种不同的访问模式，即随机访问模式和顺序访问模式。使用随机访问模式，任意给出一个网页的标识即可读取相应的网页；使用顺序访问模式，顺序读取全部网页或者一部分网页。随机访问模式主要为搜索系统的查询系统中网页缓存部分提供随机的读取需要（在第六章中还会提到）；顺

序访问模式主要用在索引系统中顺序读取网页按序索引的过程中。

3. 大规模更新（Large bulk updates）

由于万维网变化瞬息万变，所以网页库必须能够在网页删除后删除老版本的网页，如此处理可能会留下存储空洞。更新可以理解为删除后添加，而添加使用顺序添加到网页库的方法，这样不得不采用一些磁盘空间紧缩（space compaction）技术回收那些存储空洞。此外更新和访问还要保证互斥，避免同步出现的错误。

为了照顾这些特性，网页的存储方式大致分为以下 3 种类型。

（1） 日志结构（Log-structured）。

（2） 基于哈希的结构（Hash-based）。

（3） 哈希日志（Hashed-log）。

日志结构将磁盘看做一个连续存储的介质（可以想像为磁带机），这种存储方式只能顺序读及顺序写，因此存储介质对网页增加的操作非常有利。然而对于随机访问，则必须通过 B-+树的索引方式，因此每次访问需要执行两次读操作，分别读取索引和目标网页。此外还需要维护索引的代价，因此日志结构不能理想地支持随机访问的需要。

基于哈希的结构将磁盘看做一个哈希桶（Hash bucket）的集合，每个集合都足以存放在内存中。由于不需要构建额外索引，因此对于随机访问非常有利。然而对于顺序读写将十分不利，因为由于哈希函数的不确定性，对于一次网页增加，往往读入内存一个哈希桶。然后写入新增的部分，再写入磁盘。对于那些哈希桶中不变的数据，这样的一次读写操作无疑是浪费的，不断新增的网页被哈希映射到同一个桶的概率极低。而且由于哈希函数的不确定性，这种频繁读入哈希桶、更新并写入的操作被不断重复，因此对于网页新增是极其不利的。

哈希日志的结构是将哈希和日志的优势相结合，哈希作为索引找到文件块后再顺序写入，如图 3-17 所示。

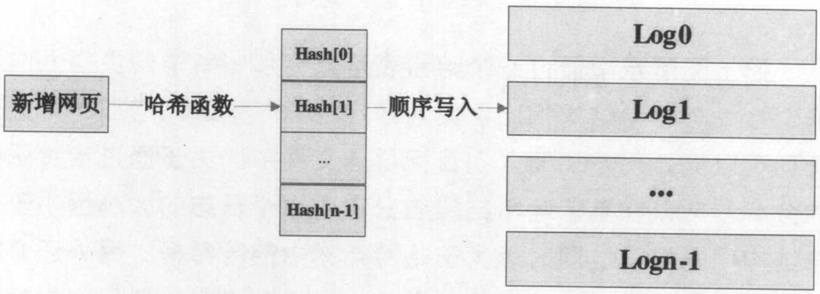


图 3-17　哈希日志的网页存储结构

对于新增的网页，通过 URL 计算出该网页的签名。进而通过模运算之后，将一个网页映射到哈希表的一个单元上，每个哈希表的单元对应于一个日志文件的位置。这个新增网页通过哈希函数的计算映射到 Hash[1]上，继而顺序写入 Log1 这个文件中。如果要随机读取某个 URL 的已存储的网页，或者仍然通过类似的哈希函数计算映射到具体的日志文件上。然后通过读取该日志文件上的 B-树索引，进而读取相应的网页文件。因此可以取得与日志

文件等价，甚至稍好的随机访问效果（随机访问的目标文件大大减小）。最值得一提的是，哈希日志的方式可以支持批处理写入，从而大大改进了纯哈希结构。在每个日志文件中增加一个写入队列，只有积累一定数量的文件后，才批量写入，如图 3-18 所示为增加排队机制的批量写入方法。

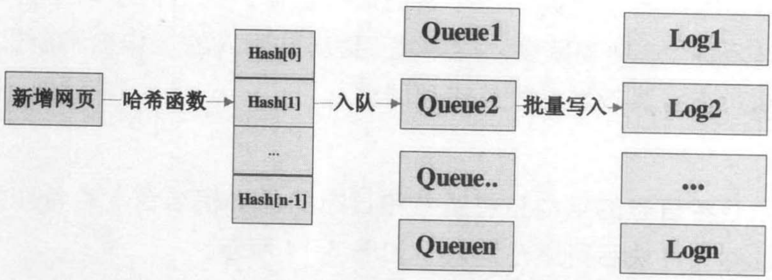


图 3-18　哈希日志批量写入记录

如上图所示，通过一个哈希表可以将那些新增网页插入的不确定性（与哈希结构相比）变为确定性插入，因此增加一个插入队列可以批处理方式插入到目标日志文件中。由于通过哈希函数的分解，所以在基于哈希结构的日志中每个日志的大小远小于日志结构中的日志，同时远大于哈希结构中的哈希桶。另外还必须保证每个日志也能足够存放在内存中，因此确定哈希日志中哈希表的大小在很大程度上还需要考虑实际物理内存大小和需要存储的网页规模。

文献[Arvind Arasu et al. 2001]中给出了对于这 3 种网页存储方式的一个定性评价，如表 3-1 所示。

表 3-1　3 种网页存储方式的一个定性评价

	日志结构	基于哈希的结构	哈希日志
顺序访问	+ +	-	+
随机访问	+ -	+ +	+ -
增加网页	+ +	-	+

只要不存在大量的随机访问机会，日志结构被认为是最佳的网页存储方式。对于可能存在大量的随机访问，以及大量新增网页的需要，哈希日志被认为是较为理想的网页存储方式。此外哈希日志能够有效地支持分布式的网页存储，在更大规模的环境中采用多机存储网页的情况下，它能够有效地将网页存储分布到每个存储结点上，增加网页存储的可靠性和稳定性。即便在某个存储结点上出现异常，对整体的搜索效果而言也不会产生较大的影响。

综上所述，网页库是搜索引擎中重要的一环，网页库具有不同的访问需求（顺序访问及随机访问），具体问题具体分析时需要结合这些需求选择相应的存储方式。在实践中，日志结构和哈希日志是较为常见的选择。

03 第六节 下载系统回顾及未来发展

通过系统地学习，至此终于揭开了搜索引擎下载系统的神秘面纱。实现下载系统的主要需求，总结下来最主要是以下 3 点。

（1）**抓得全**：通过网页更新策略（更新目录型网页）。

（2）**抓得快**：通过合作抓取策略。

（3）**代价低**：通过宽度优先的遍历策略、最大深度策略及合理的网页更新策略。

当然这与一个实际的商用大型搜索引擎下载系统还有很大距离，例如大规模网页存储、DNS 服务模块、反垃圾、反病毒、多爬虫的协调工作，以及爬虫监控等。读者如果亲自实现一个爬虫，还会遇到很多困难。目前比较著名的关于爬虫的开源代码很多，例如 Web sphinx 和 Heritrix 等。读者可以在学习本章内容后，实际参与到这些开源代码的学习和使用中，以进一步加深对搜索引擎的下载系统的认识。

虽然前人做出了巨大的努力，取得了大量的成果，然而爬虫的发展还远远没有停止。未来还有很多工作等待我们去探索，那么还有哪些工作可以做呢？

1. 动态网页支持

Web 上动态网页是静态网页的 400~500 倍，光明星球[brightP]公司宣称，存在的网页总数至少为 5 500 亿个，这个数量是相当惊人的。目前几乎所有的搜索引擎都不能完全解决抓取动态网页的难题，因为这些动态网页通常都受到了账号和密码的保护，这也称为“深度挖掘问题”。

2. 定向抓取

定向抓取通常也称为“聚焦爬虫”，目的是使爬虫的工作方式不再是漫无目的，而是在某种意图下抓取有价值且特定的网页。

3. 抓取 XML 网页，以及 DOC 和 PDF 文件

抓取的内容突破 HTML 网页，也要抓取分析那些 XML 网页，以及 DOC 和 PDF 等文件。

4. 智能分布抓取

随着 Web 的迅猛发展，可以考虑物理上分布多个可扩展 Web 搜集系统，在更广泛意义下以并行分布式工作。

读书笔记

参考文献

[1] [Reka Albert et al. 1999] Reka Albert, Hawoong Jeong and Albert-Laszlo Barabasi. Diameter of the World-Wide Web. Nature 401, 130-131, Sep. 1999

[2] [Arvind Arasu et al. 2001] Arasu, A. and Cho, J. and Garcia-Molina, H. and Paepcke, A. and Raghavan, S. Searching the Web, ACM Transactions on Internet Technology,42 pages

[3] [Bloom, B.H. 1970] Space/time trade-offs in hash coding with allowable errors Communications of the ACM, 1970

[4] [BrightP]TheDeepWeb: Surfacing Hidden Value. http://www.brightplanet.com/deepcontent/tutorials/DeepWeb/index.asp

[5] [Broder, et al.,2000] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and a. J. Wiener, "Graph structure in the web: experiments and models," presented at Proceedings of the 9th World-Wide Web Conference, Amsterdam, 2000

[6] [BRIN, S. AND PAGE, L. 1998] The anatomy of a large-scale hypertextual Web search engine.Comput. Netw

[7] [CHO, J. AND GARCIA-MOLINA, H. 2000a] Estimating

frequency of change. ACM Transactions on Internet Technology, Vol. 3, No. 3, August 2003

[8] [CHO, J. AND GARCIA-MOLINA, H. 2000b]. Synchronizing a database to improve freshness. In Proceedings of the ACM SIGMOD Conference on Management of Data. ACM Press

[9] [CHO, J. AND GARCIA-MOLINA, H. 2000c] The evolution of the web and implications for an incremental crawler. In Proceedings of the 26th International Conference on Very Large Data Bases

[10] [HIRAI et al 2000]HIRAI, J., RAGHAVAN, S., GARCIA-MOLINA, H., AND PAEPCKE, A. 2000. Webbase: A repository of web pages. In Proceedings of the Ninth International Conference on The World Wide Web. 277–293

[11] [Jeff Heaton,2002] Programming Spiders, Bots, and Aggregators in Java Jeff Heaton Publisher: Sybex , February 2002 , ISBN: 0782140408, 512 pages

[12] [Li02] 李晓明，"对中国曾有过静态网页数的一种估计"，PKU_CS_NET_TR2002006, 2002 年 4 月

[13] [闫宏飞,2002] 闫宏飞，"可扩展 Web 信息搜集系统的设计、实现与应用初探" 北京大学,博士论文，2002, pp. 116

[14] [闫宏飞 and 李晓明,2002] 闫宏飞，李晓明，"关于中国 Web 的大小、形状和结构" 计算机研究与发展，vol. 39，No.8，2002

第四章

搜索引擎的分析系统

第一节 知识准备 04

搜索引擎的 4 大系统中的第 2 个系统是分析系统，分析系统主要完成的工作包括信息抽取、网页消重、中文分词和 PageRank 计算等。接下来的各节将按照这个顺序层层深入地介绍分析系统的工作，在此之前了解一些基本概念。

HTML 语言

HTML 语言（Hyper Text Markup Language，超文本标记语言）是一种专门的编程语言，用于创建存储在 WWW 服务器上的文件，并能由 Netscape 及 Microsoft Explorer 等浏览器浏览。由于 HTML 语言简单易用，不需要编译的特性，因此被广泛地使用。

锚文本（anchor text）

锚文本网页中关于链接的一段描述，通常以文本和图片的方式出现。可以指向文中的某个位置，也可以指向其他网页。例如，

HTML 脚本走进搜索引擎中的"走进搜索引擎"就是一个"锚文本"，它用来描述其指向的链接。

半结构化数据（Semi-structured data）

和普通纯文本相比，万维网上的网页数据具有一定的结构性，但是这种 HTML 标签带来的结构性不能满足网页结构化的需要。例如<TITLE>标签标识网页主题，而<TD>标签有些表示文章主题，有些表示文章段落，或者其他广告信息等。因此人们称网页的原始数据为"半结构化数据"，这是 HTML 语言的基本特点。

第二节 信息抽取及网页信息结构化 04

对于分析系统来说，基础和首要的工作是分门别类地从半结构化网页（semistructured page）中抽取（extract）出有价值的能够代表网页的属性，例如锚文本、标题和正文等。并将这些属性结合起来组成一个网页对象，这种处理称为“网页结构化”。

网页结构化的目标

网页结构化的目标是针对搜索的需要，将半结构化的 HTML 网页中的数据按照如下几个基本属性依次抽取，最后打包（wrap）出一个网页对象。

（1） 锚文本（anchor text）。

（2） 标题（title）。

（3） 正文标题（content title）。

（4） 正文（content）。

（5） 正向链接（link）。

网页的这5个基本属性对于检索来说至关重要，下面一一介绍。

（1） **锚文本**除了网页标题可以描述网页以外，还会有一些锚文本来描述它。例如清华大学主页可能被另外一些网页中存在锚（anchor）所指向，其锚文本就是该网页的最佳描述。特别是对于某些没有标题的网页，锚文本是有益的补充。

（2）**标题** :这里的标题特指HTML标识语言中<title></title>中间的文字部分，这部分文字表达了网页的基本含义。和锚文本相同的是，都是用来描述网页的内容的属性；和锚文本不同的是，这个标题是由该网页制作者本人编写的。

（3） **正文标题**：在HTML网页中，网页的标题由<title>标签标识。实际的情况是由于网页编写者的疏忽，或者其他原因，<title>标签中的文字没有任何网页描述能力。因而并不是合格的标题，为此需要抽取正文中的适当文字作为正文标题。

（4） **正文**：锚文本、标题和正文标题都是网页的简短描述，而正文是一个网页的主体内容，它完整地表述了网页的主体内容，一般出现在<DIV>、<TABLE>和<P>等HTML标签中。

（5） **正向链接**：正向链接是网页制作者编写的引导用户继

续在网上冲浪的链接，这些链接的文字也是其他网页的锚文本。

因此一个网页对象（Page Object）简单说来至少包含 5 个属性，其含义如表 4-1 所示。

表 4-1　网页对象的 5 个属性及其含义

属性	含　义
title	页面标题
content title	正文标题
content	正文
link	正向链接集合
anchor	指向自身的锚文本

在不失完整性的前提下，不妨对问题进行简化，通过结构化网页中标题和正文两种网页对象的属性来理解网页结构化的基本原理。

假定下面这个简单 HTML 网页的文件名为“book.html”：

```
<HTML>
<HEAD><TITLE>走进搜索引擎</TITLE> </HEAD>
<BODY>
<TABLE>
<TR><TD>走进搜索引擎：第一章</TD> </TR>
<TR><TD>走进搜索引擎：第二章</TD> </TR>
<TR><TD>走进搜索引擎：第三章</TD> </TR>
</TABLE>
```

```
</BODY>
</HTML>
```

将以上内容写入文件 index.html，用 IE 浏览器打开，显示如图 4-1 所示。

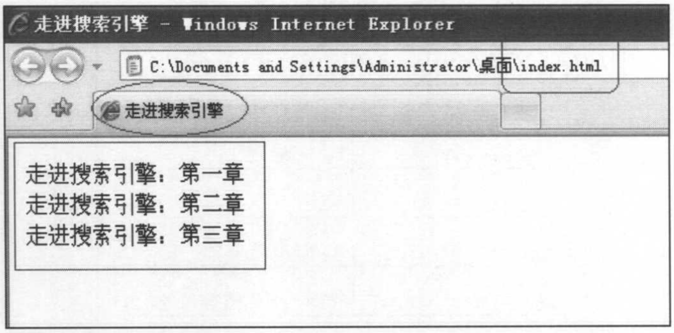


图 4-1　包含标题和正文的简单网页示例

圆角矩形框表示该网页的文件名；椭圆框表示网页的标题（TITLE）；矩形框表示该网页正文，其中包含几段文字。

网页结构化的过程可以直观地理解为如图 4-2 所示的过程。

在理解结构化网页的目标后，加上对 HTML 语法的特点，以及搜索引擎的实际需要，为这种分析原始网页的过程制定了如下“两步走”的方法。

（1） 建立 HTML 标签树（tag-tree）。

（2） 通过投票方法识别正文的文本块，并按照深度优先遍历的规则组织为正文。

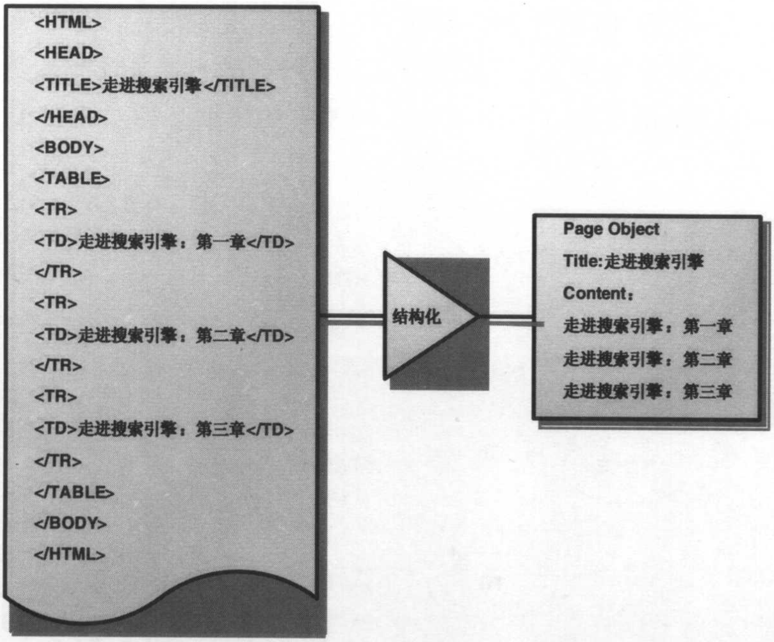


图 4-2　网页结构化的过程

建立 HTML 标签树

万维网上大多数的静态网页都以 HTML 网页形式存在，HTML 是一种标识语言（Markup Language），它把其描述的全部内容都按照 HTML 语法存放在标签之中。为了更清楚地描述网页内容的组织结构，将网页中的标签按照出现顺序依次整理出来并用适当的结构记录。由于标签之间的嵌套关系，因此整理结果自然是一个树状结构，我们把整理一个网页中的标签得到的树状结构称为该网页的“标签树”。

很明显，浏览该网页的用户看到的是相当友好的信息。而实际源文件中的那些 HTML 标记，如<HTML>和<TABLE>（可以理

解为用来帮助 IE 浏览器理解网页）等都不会实际地展示给用户。因此分析系统需要学习 IE 浏览器理解网页的方式来理解网页，在理解过程中需要建立一个 HTML 标签树的树型结构，如图 4-3 所示。

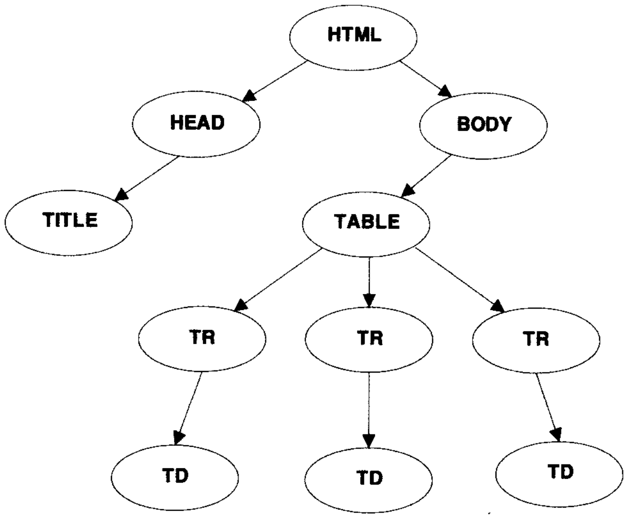


图 4-3　HTML 标签树

HTML 语法中每种标签都是成对出现的，这有助于标记一个标签的始末，因此需要一种能够记忆历史的数据结构来协助标签树创建和分析过程。下面通过一个例子来理解这个标签树建立的过程。

首先，该过程需要一个标签分析栈的数据结构。栈结构是一种先进后出的线性表结构，栈结构的这种特性为分析工作提供了可能。通过简单的图形可以把实际的标签抽取工作容易地表达出来，如图 4-4 所示。

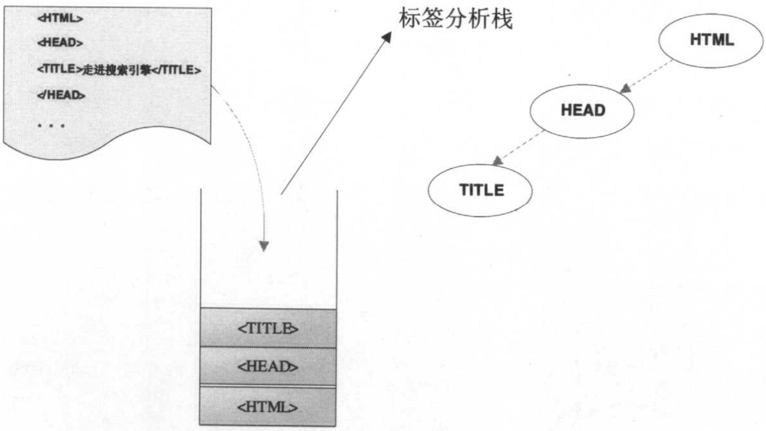


图 4-4　标签分析栈

文本读取程序将网页源文件中的<HTML>、<HEAD>和<TITLE>依次投入分析栈中，在右边依次建立一个标签树，栈底元素<HTML>为树根。当读取到“走进搜索引擎”时发现该文本不是标签，于是将其保留。由于栈顶标签为 TITLE，因此该字符串为正文标题，如图 4-5 所示。

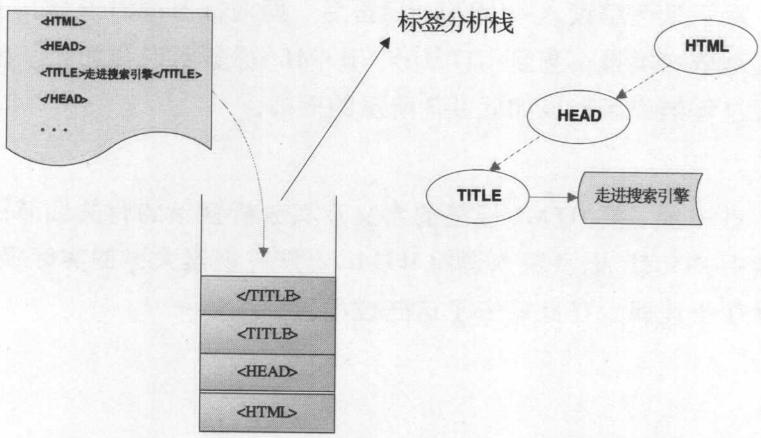


图 4-5　抽取 TITLE 标签所含文字

读书笔记

文本读取程序继续读出</TITLE>这个标签，并将其放入标签分析栈中。此时，标签栈顶的标签为<TITLE>，因此这两个成对标签同时退栈，如图 4-6 所示。同时也确认了“走进搜索引擎”这一字符串为正文标题，在标签树上的 TITLE 标签存放一个指向该字符串的指针，这样 TITLE 标签的文字被正确地抽取出来。

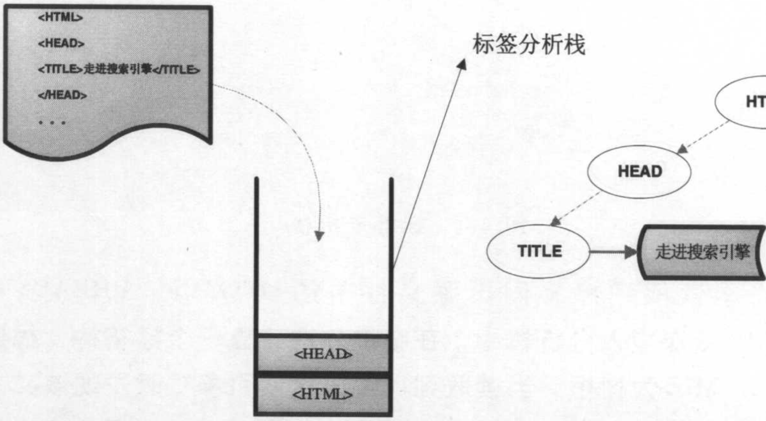


图 4-6　退栈过程

接下来继续读入</HEAD>标签后，两对标签同时退栈。下面的过程依次类推，直到<HTML></HTML>标签对退栈为止，整个分析过程结束后形成如图 4-7 所示的形态。

进一步了解 HTML 标签的含义及其分析容错的有关细节需要阅读相关文献，以及深入理解 HTML 语法并积累大量的实际经验，为了便于理解，在此略去了这些过程。

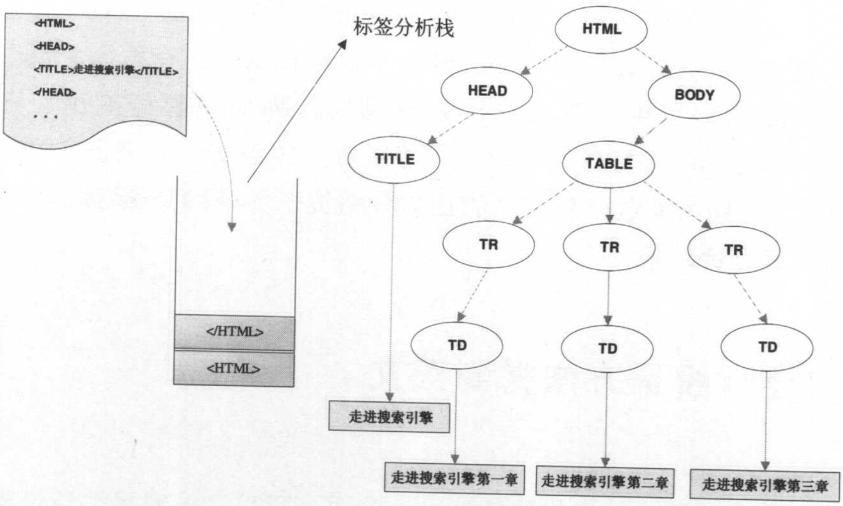


图 4-7　分析过程结束后形成的形态

通过建立标签树，并且识别标签所描述的文字，网页结构化的进程走出了重要的一步，如图 4-8 所示。

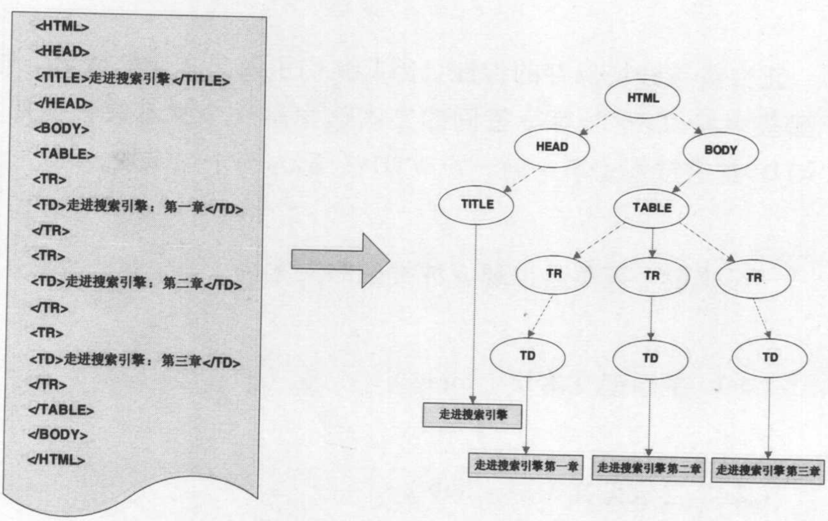


图 4-8　从 HTML 网页到结构化网页

通过上面这个过程已经能够提取出标题，很明显 TITLE 标签下的文字即为标题。进展令人振奋，然而离目标还有很大距离。在实际的网页中，正文、广告及其他各种信息都可能出现在<table>和<td>等标签中，哪一个才是真正的正文？由于正文可能分片，例如上面这个例子中的正文分别位于 3 个<TD>标签上，如何将其组合起来？

通过投票方法得到正文

由于网页半结构化的特性，一方面，TITLE 很容易地通过第 1 步标签分析得到；另一方面，得到完整的正文却非常复杂。首先，网页中没有明显的标签标识出正文；其次正文可能分散在多个 HTML 标签中，问题在于如何组合出完整的正文。首先来解决第 1 个难点，即识别正文。

正文具有分块保存的特性，因此我们引入文本块的概念，对于那些诸如<P></P>等标签间的文本认为是一个文本块。例如，"<TD>走进搜索引擎：第一章</TD>"称为一个文本块。

一般来说，网页会出现 3 种类型的文本块。

（1） 主题型文本块（topic）。

（2） 目录型文本块（hub）。

（3） 图片型文本块（pic）。

主题型文本块是大段文字的文本块，例如：“<TD>走进搜索引擎：第一章</TD>”。

目录型文本块是描述链接的文本块，例如：“走进搜索引擎：第一章”。

图片型文本块是描述图片的文本块，例如：“<img src="">走进搜索引擎：第一章</img>”。

目录型文本块和图片型文本块相对容易被区分；而主题型文本块中可能包含广告等其他内容，必须与正文相区别。

判断哪个文本块是正文采用称为“投票算法”的计算方法，这种方法在搜索引擎中特别常用。

在日常生活中几乎所有人都会有投票或者选举的经历，例如选举干部和通过决议需要投票，以及运动员的一套动作需要裁判员打分等。其基本原理在于认为大多数人的意见往往是正确的，大多数人的统一主观意见就会变得较为客观。例如在体操运动员结束一套动作之后，多个裁判根据自己主观意见给出独立的评判。然后去掉一个最高分和一个最低分，这时的分数尽管是主观产生的，但是这种评判的方法和结果被认为是相对客观和可信的。

正文抽取的投票算法的过程如何呢？首先我们会定义一系列规则，然后通过这些规则为每一个文本块打分。得分最高的被认为是正文的可能性足够大，并且可以接受。

我们来举这样一个例子，假定一个规则集合中包含以下 3 条规则。

（1） 如果文本块文本的长度少于 10 个字，得分为 0；介于 10 个~50 个字得分为 5 分；介于 50 个~250 个字，得分为 8 分；超过 250 个字，得分为 10 分。

（2） 如果文本块文本位置在右侧，得分为 0 分；在顶部，得分为 3 分；在左侧，得分为 5 分；在中间，得分为 10 分。

实际的规则往往更加繁多和复杂，为了介绍该方法，我们定义文本长度和文本位置规则。投票算法的过程是依据不同的规则从不同的角度依次打分，文本块得分高的是正文的一部分。投票算法的一般模型如图 4-9 所示。

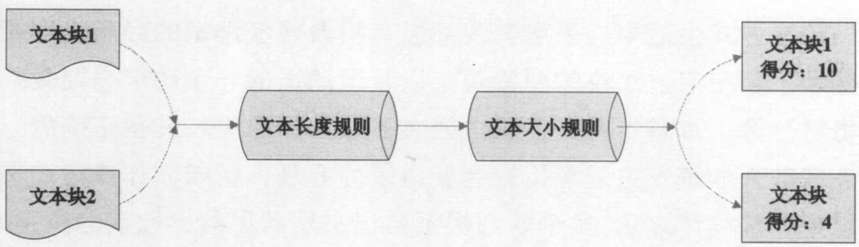


图 4-9 投票算法的一般模型

如图所示，每个规则器打分后，累积总分为文本块 1 得 10 分，远大于文本块 2 得到的 4 分，因此可以认为文本块 1 是正文内容的可信度高。

在实际的程序中，规则器的打分是不断调整的，其个数也可

读书笔记

以支持动态添加。如果规则打分计算比较复杂耗时，则可以采用并行打分的方法。这样所需要的时间将由最耗费时间的规则计算代价决定；否则通过串行打分，时间耗费将会是所有规则计算代价的总耗时。为了简化，图 4-9 中采用串行打分的方法。

除此之外，规则的定义还需要通过足够多的网页进行反馈，之后才能得到一个公正客观的打分。这类似裁判员在打分前的业务学习，只有在比赛前透彻地理解评分标准，并且很好地观察运动员完成动作的情况。通过不断地训练才能区别动作的难易程度，达到比赛裁判的水平。此外，有些动作在若干年后由于掌握的运动员很多，因此评分标准就会发生变化。新出现的高难动作将会得到评分上的青睐，规则器也需要类似这样的业务学习过程。如果经常发现某些网页的正文段抽取错误，则要找出是哪一个规则器打分不合理才会导致这个结果。反复这样的过程，最后的打分将会趋于合理。

最后还需要注意这样的问题，如果规则 A 打分为 0 分~1 000 分，规则 B 为 0 分~10 分。那么很显然，规则 B 对最后打分结果的影响力几乎可以不考虑，因此如何平衡规则对最后结果的影响力也需要充分考虑。这是因为规则的重要性不同可以适当拉开距离，但有时需要在可以接受的范围内保持这种距离。

即便做出了如此多的工作，投票的结果也并不总是正确的。真理也常常会站在少数人的一边，群众的眼睛也不可能总是雪亮的，做到百分之百的判断准确即便对人来说也都是异常困难的。因此采用这种方法永远不可能完全正确，只能是达到某种概率下的正确，而作为投票算法追求的目标是将这种得到正确结果的概率不断地提高。

剩下的工作就是如何将一个个文本块组织成一个正文？深度优先遍历标签树并依次记录主题类型的文本块，即可得到该网页的正文。为了说明深度优先遍历的方法，我们使用一个略微复杂的例子，如图 4-10 所示。

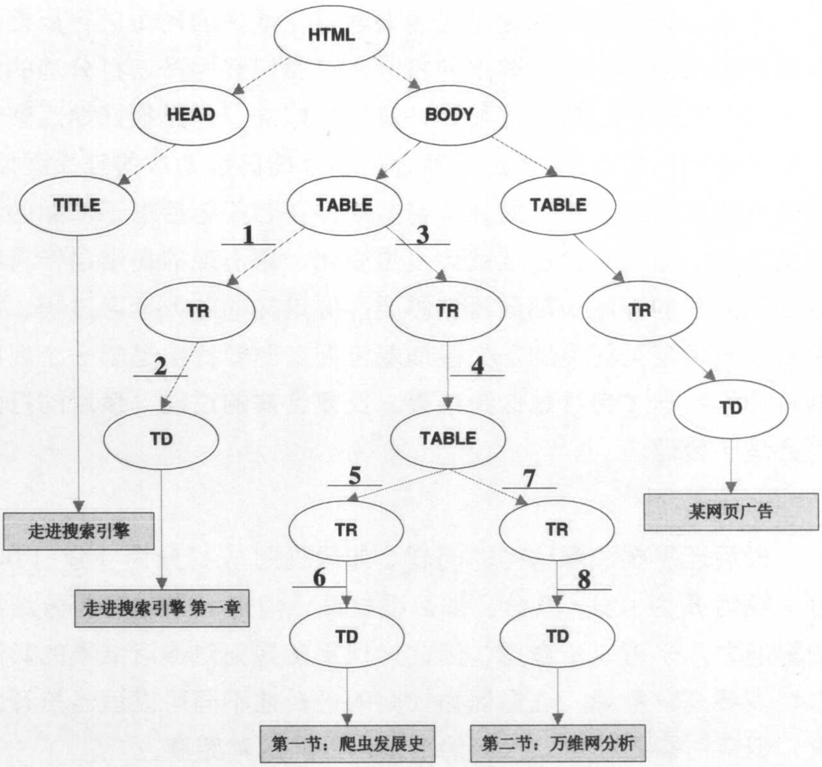


图 4-10 提取正文的方法

在图 4-10 所示例子中的 BODY 标签下，第 1 个 TABLE 为正文，因为其下的文本块通过投票打分均为文本块。而第 2 个 TABLE 标签下通过投票打分，其得分较低，因此可能为广告类信息。这样组织正文只需要从第 1 个 TABLE 开始深度优先遍历，遍历顺序为图中带下划线的数字所示，因此依次提取的文本块并按照顺

序组织成如下的正文："走进搜索引擎 第一章 第一节：爬虫发展史 第二节：万维网分析"。正是由于 HTML 标签的这种相互嵌套的特性，所以决定了深度优先遍历的顺序恰好能够组织成一个完整的正文。

对于其他的网页属性抽取，例如正文标题等也大多采用相同或类似的方法。

锚文本的提取有一些复杂，由于锚文本对网页的描述存在于其它网页中，例如南京大学的主页，被其它网页所链接，这些网页中包含了"南京大学主页"或"南京大学"等这样的锚文本，因此南京大学主页这个网页对象的锚文本就是"南京大学"、"南京大学主页"。锚文本提取，筛选等工作需要在获得足够多的网页后才能进行，采用分块计算的方法，比较复杂，这里不再展开叙述。

网页结构化过程回顾

网页结构化的过程首先通过标签树进行分析得到文本对应的标签，然后通过投票算法确定正文及配图等仅从 HTML 标签无法判断的网页数据。这样就圆满完成了结构化的任务，达到了理想的结构化要求，如图 4-11 所示为两步走过程。

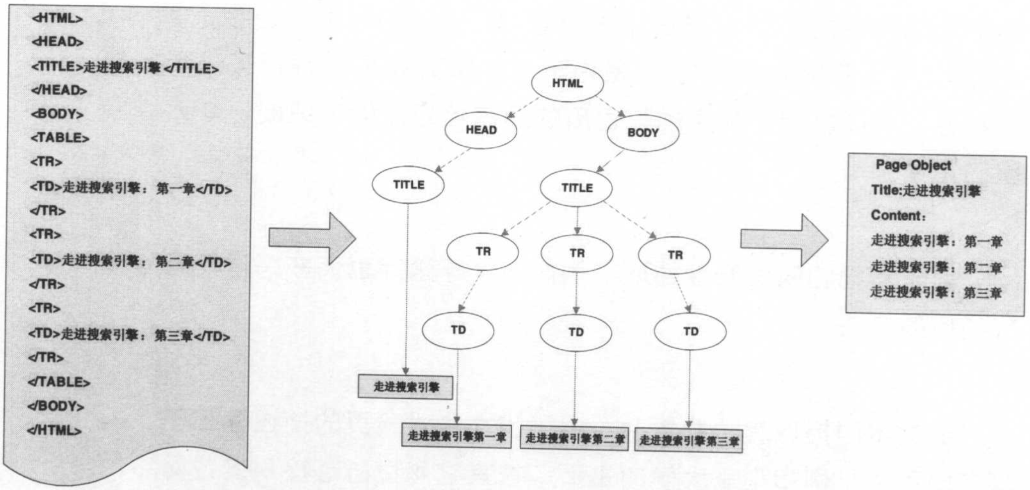


图 4-11　两步走过程

直观地感觉到网页结构化的过程使得网页有价值的信息被保留，例如标题和正文；而网页中无用的信息被丢弃，例如 HTML 标签。因此网页结构化的过程同时也节约了大量的存储，存储一个网页不再是存储原始的网页，而是只需要存储结构化的网页。即便是这样，存储和处理大规模网页依然是一项艰巨的任务，解决这个问题首先想到的是应该去掉那些相同或者类似的网页。这样不仅可以大大节约存储，还可以有效地提高检索效果，下一节将介绍有关网页查重的技巧。

第三节 网页查重 04

在日常上网时一般很少会留意到相同相似网页，一些偶然的机会也能看到一些相对重要的网页会被各大网站转载，或者在浏览 BBS 时会看到各种各样的转帖。对于网民来说，这种重复是有利的，便于大多数人看到重要的信息。然而对于搜索引擎来说，重复网页的存在意味着这些网页至少被多处理一次。更糟糕的是，在接下来的索引制作中可能会在索引库中索引两份相同的网页。当有用户查询时，在有限的查询结果页中（一个查询结果页显示 20 条网页链接）就会出现重复的网页链接。因此无论从系统效率，还是从检索质量来说，重复网页都是有害的。

网页查重技术发展历史

网页查重技术起源于复制检测技术，复制检测即判断一个文件的内容是否抄袭、剽窃或者复制于另外一个或者多个文件。

1993 年，Arizona 大学的 Manber 推出一个 sif 工具，用于在大规模文件系统中寻找内容相似的文件。1995 年，Stanford 大学

的 Brin 和 Garcia-Molina 等人在“数字图书馆”工程中首次提出了文本复制检测机制 COPS（Copy Protection System）系统与相应算法[Sergey Brin et al 1995]。之后这种检测重复技术被应用到搜索引擎中，基本的核心技术即比较相似。

和简单的文档不同，对于网页来说，这种相似包含了特别丰富的含义。网页具有内容和格式，因此在内容和格式上的相同相似构成了如下 4 种网页相似的类型（如果把相同看做相似的一种特殊情况）。

（1） 两个网页的内容和格式上完全相同（full-layout duplicates）。

（2） 两个网页的内容完全相同，但格式不同（full-content duplicates）。

（3） 两个网页有部分重要的内容相同并且格式相同（partial-layout duplicates）。

（4） 两个网页有部分重要的内容相同，但格式不同（partial-content duplicates）。

如图 4-12 所示，搜索引擎在相似问题上按照内容和格式的组合共计 4 种不同的形式。对大规模网页来说，网页查重只需要找到内容相同或相似的网页，而忽视格式上的异同，因此在网页查重的过程中，图 4-12 所示的 4 种相似情况都属于网页查重的目标。

采用网页签名的方法很容易检测网页完全一致。如果两个网页签名值相同，则为相同网页，这种方法在第三章中关于网页

URL 不重复抓取的策略中已介绍过。因此本节主要关注的是检测网页相似的情况。

检测大规模的相似网页需要考虑准确性和效率，文献[Abdur chowdhury et al 2002]，[Broder 1997]分别提出了两种经典的处理大规模网页相似检测的方法，下一节我们将看到这一有趣的方法。

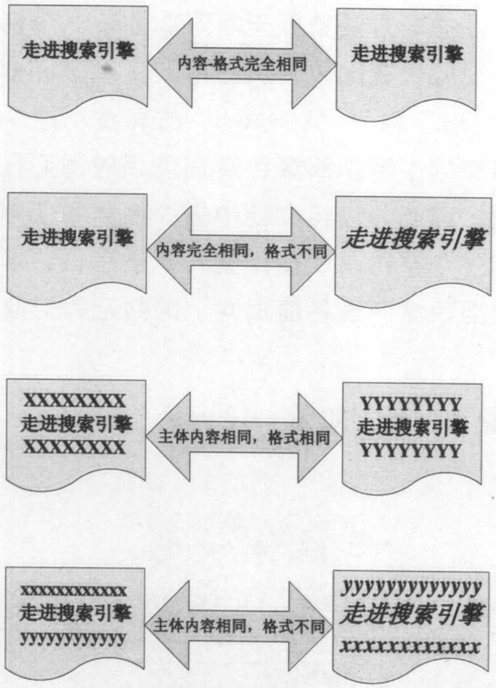


图 4-12　网页相似性的 4 种形式

网页查重实现方法

由于不考虑格式的异同，网页查重首先需提取结构化网页的

读书笔记

正文和标题，将复杂的网页转化为具有标题和正文的文档，因此下面称网页查重为“文档查重”。

日常生活中判断两个相似事物一般采用比对特征的方法，特征是那些有代表性、不易变化、能够反应主体本质的信息。文档查重的第 1 步就是特征抽取，下面介绍两种特征抽取的方法。

第 1 种特征抽取方法着眼于尽可能抽取一个特征，这样比较两个文档是否相似只要比较一次即可。文献[Abdur chowdhury et al 2002]首先提出了称为“I-Match”的算法，这个算法基于一个假设，即一篇文档中特别高频和特别低频的词汇往往不能反映这篇文档的本质。因此通过将文档中去掉高频和低频词汇后的词汇通过排序得到一个字符串，使用签名算法获得该字符串的签名。如果有其他文档和这个签名值相同，则判定为相似。

I-Match 算法的示例如图 4-13 所示。

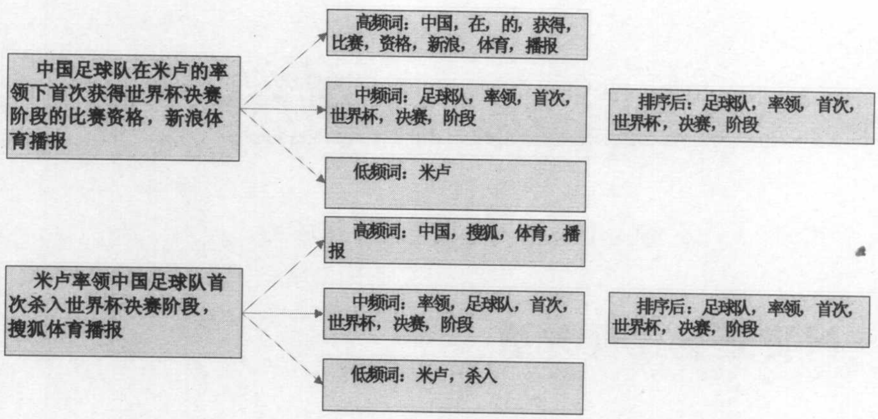


图 4-13 I-Match 算法抽取特征

如图 4-13 所示的例子中，通过去掉高频和低频词汇（词频的统计工作由索引系统提供，在第五章中会提到词频的统计方法和技巧），两个文档趋于一致，通过排序则得到完全相同的词汇列表。通过这种计算，两篇文档的相似性就转化为一个词汇列表是否相同的问题。

第 2 种特征抽取方法是抽取多个特征词汇，通过比较两个特征集合的相似度实现文档查重。抽取多个特征的方法很多，这里介绍其中的一个经典算法，称为“Shingle 算法”。Shingle 在英文中表示相互覆盖的瓦片。接下来通过上面的例子来说明 Shingle 算法为什么命名为“Shingle”。例中使用每 5 个汉字为一个 Shingle 的方法，如图 4-14 所示。

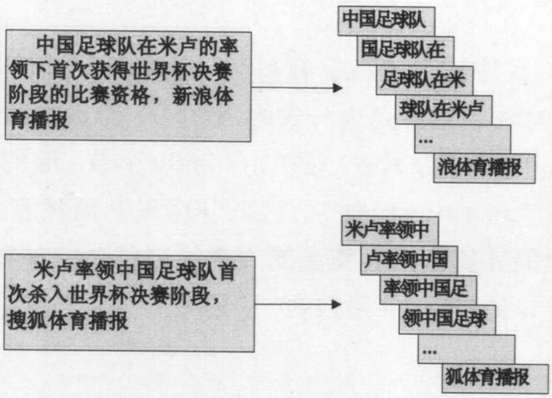


图 4-14　Shingle 算法抽取特征

如图所示，每 5 个汉字组成的 Shingle 恰如其名，好似相互覆盖的瓦片一样。由于把一个文档转化为一组字符串集合（每个元素为一个 Shingle），因此判断两个文档的相似性就转化为字符串集合的相似性。

最后介绍一个关于 Shingle 算法的简单结论，对于长度 L 的文档，每隔 N 个汉字取一个 Shingle，这样一共取到 L - N + 1 个 Shingle，可见 N 的取值对效率和效果的影响都很大。很显然 N 最小取 2，最大取 L。

在特征抽取完毕后，就需要进行特征比对，因此网页查重的第 2 步就是相似度计算和评价。

I-Match 算法抽取的特征只有一个，因此只需要为文档计算一个签名值，如果两个文档的签名值相同，则文档相似。大规模文档做查重只需要维护一个哈希表即可，每来一个文档查找一次哈希表。如果哈希表的槽位（bucket）被置位，说明已经存在相似文档。

Shingle 算法抽取的特征有多个，因此处理起来稍微麻烦一些，比较的方法是记录完全一致的 Shingle 个数。然后除以两个文档的 Shingle 总个数减去一致的 Shingle 个数，这种方法计算出的数值称为“Jaccard 系数”，它可以用来判断集合的相似度。Jaccard 系数的计算方法为集合的交集除以集合的并集，即若令集合 A 与集合 B 的 jaccard 系数为 J，则有：

$$J = \frac{A \cap B}{A \cup B}$$

在如图 4-14 所示的例子中，完全一致的 Shingle 分别是“中国足球队”、“世界杯决赛”、“界杯决赛阶”和“杯决赛阶段”。其中第 1 个文档具有 Shingle 总数是 30 个（计算方法为正文总长度 -Shingle 长度+1），第 2 个文档 Shingle 总数为 22 个，因此两个文

档的相似度（jaccard 系数）为 $\frac{3}{30+22-3}=0.06$。在通过 jaccard 系数量化了相似度后，即可通过评价这个数值是否达到相似所需要的标准来判断是否相似。如果在搜索引擎中相似度达到 0.2 才能判断为相似，那么这个例子的相似度没有达到标准，因而不能看做是相似文档。

对比一下这两种方法，I-Match 算法提取特征时需要文本分词和词频比较的代价，因此提取特征比较复杂，但文档是否相似的计算简单；Shingle 算法提取特征简单，但文档是否相似的计算复杂，因此各有利弊。由于 Shingle 算法在性能上的优异表现，因而被广泛采用。

这里介绍的是最经典的两种方法，其他网页查重的方法还有很多。文献[李晓明 2004]中介绍了多种网页查重的方法，有兴趣的读者可以深入阅读。

网页查重的目标是消重，因此网页查重的第 3 步就是网页消重。

消重就不可避免遇到这样一个问题，即在相同或者相似的网页集合中保留哪一个，而消除哪些呢？从版权的角度考虑，应该尊重原创，过滤转载或者复制的网页；从网页寿命的角度考虑，过滤掉那些网站质量不高的网页，保留大型网站的网页；从容易实现的角度考虑，首先保留被爬虫抓取的网页，然后丢弃被抓取的相同或相似网页。最后一种方法最为简单实用，由于保留先被爬虫抓取的网页同时很大程度上也保证了优先保留原创网页的原则，因此被广泛采用。

综上所述，一般说来，网页查重至少需要如下 3 个主要步骤：

（1） 特征抽取。

（2） 相似度计算、评价是否相似。

（3） 消重。

网页查重工作是分析系统不可缺少的重要一环，由于去除了重复网页，所以在接下来的其他模块的处理环节中，节约 PageRank 的计算代价、节省了索引存储空间、减少了查询成本、提高了查询结果的多样化效果。

第四节 中文分词 04

网页查重的工作完成后，分析系统在将分析的结果发往索引系统前还需要对正文进行分词，也称为“切词”（word segment）。分词的方法繁多，本节介绍一些基本方法。

什么是中文分词

任何文档都可以看做是一些连续的词的集合，然而中文并没有明显的词间分隔，这一点和英文不同。当然英文也有难点，例如时态和词性的变化等。在中文语法中，词汇是由两个或两个以上汉字组成的，并且句子是连续书写的，句子间由标点分隔。这就要求在自动分析中文文本前，首先将整句切割成小的词汇单元，即中文分词。

举个简单的例子来说明分词的难度，对于“学历史学好”这个句子，作为人来说，能够很容易得到正确的切分方法，即“学/历史学/好”（本书用“/”表示一个分词切分符号）。然而计算机要具有这样的智慧还有很多工作要做，否则可能分为“学历/史学

/好"。

目前的分词手段主要依靠了字典和统计学的方法。由于索引是按照关键词建索引的，所以分词的效果直接决定了索引词及检索的效果。因此例如将文档"学历史学好"错分成"学历/史学/好"，所以索引时，只会对"学历"、"史学"及"好"这3个索引词建立它们与该文档的关联关系。这样用户查询"历史学"这个关键词时，就无法检索出这个文档，可见分词质量在很大程度上影响了搜索的结果和效果。

通过字典实现分词

先人为我们沉淀了厚重的历史和灿烂的文化，而其中的代表就是我们日常使用的文字。也许正是因为没有断词，所以它才显得那么奇妙和多变。虽然这种多变在表达上提供了灵活性，但给搜索引擎带来了难题。在中文分词中主要体现了如下3种难分类型。

1. 交集型歧义

在阿拉伯数字字串AJB中，若AJ∈D、JB∈D、A∈D且B∈D，则AJB为交集型歧义字段。此时，AJB有AJ/B和A/JB两种切分形式，其中J为交集字段。例如，"从小学"，这个词可能有多种切分方法。

对于"从小学电脑"，正确的切分为"从小/学/电脑"。

读书笔记

对于“从小学毕业”，正确的切分为“从/小学/毕业”。

2. 组合型歧义

在字串 AB 中，若 AB∈D、A∈D 且 B∈D，则 AB 为组合型歧义字段。此时，AB 有 AB 和 A/B 两种切分形式。

“中将”这个词可能有多种切分方法。

对于“美军中将竟公然说”，正确的切分为“美军/中将/竟公然说”。

对于“新建地铁中将禁止商业摊点”，正确的切分为“新建地铁/中/将/禁止商业摊点”。

3. 混合型歧义

同时包含交集型歧义和组合型歧义，则为混合型歧义。

对于“人才能”，可能切分为“人才/能”、“人/才能”和“人/才/能”。

目前解决这些问题的技术手段主要借助字典和统计学的方法共同完成。我们首先从字典开始讲起，梁南元教授[梁南元，1987]最早提出利用字典的方法分词。但如何让计算机读字典，如何让

读书笔记

计算机在各种字典的词义中选择合理的分词却并不容易。

日常使用的字典通常收录那些意义明确、使用频率高且生活中约定俗成的词汇，对于搜索引擎来说，字典中收录的词汇相当少。但即便是如此少的词，如何很快并有效地存储字典，以至于在需要知道一个字符串包含多少个字典中出现的词时能够快速地实现进行过程，成为首先需要解决的问题。

字典通常采用前缀树或者后缀树的数据结构存储，什么是前缀树？我们来举个例子。

将字典做成一个前缀树结构的数据结构，如图 4-15 所示。

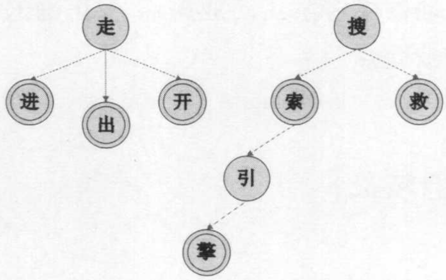


图 4-15 前缀树结构的词典组织形式

选择若干词建立一个前缀树数据结构作为示例，整个字典将是一个由多个后缀树构成的森林。其中从树顶开始，每一个字及其全部祖先构成一个词汇的一部分。对于图中双线圈的词，认为是一次终结，表示它及其全部祖先可以构成字典中的一个词汇。这里“走进”是一个词汇，而“搜索引”不是一个词，因为“引”是单线圈。“搜索引擎”是一个词汇，因为“擎”这个字是双线圈。这种结构十分类似于日常生活中字典的组织方式，即均以一个字

开头，其后为以该字开头的全部词汇。日常生活中查字典的习惯也是如此，在字典中查一个词。首先查出该词汇的首个汉字在字典中的页数（这里可以理解为在字典森林中找到以某个字开头的前缀树），继而在其中查找相应的词汇（可以理解为在该词的后缀树中查到指定的词汇）。

分词的过程可以理解为一次查字典的过程，假定有一个句子“走进搜索引擎。”需要分词，那么计算机将会如何切分呢？

首先计算机读入“走”，找到“走”这个字典分支。然后读到“进”，查看“走”字打头的前缀树。发现“进”是双圈字，与“走”恰好构成一个词。继续读到“搜”时，发现“走”字开头的前缀树中没有继续的匹配，因此切分出“走进”这个词汇。

接下来找到“搜”这个字典分支，继而读入“索”。发现“索”是一个双圈字。系统确定了一个词“搜索”，然而还可能存在更长的匹配。此时系统继续读入“引”，由于这时“引”是单圈字，因此继续向下读入“擎”。发现“擎”为一个双圈字，由于下面继续读出的是句号，因此抽取出“搜索引擎”这个词汇。这样整个句子切分完毕，结果为“走进/搜索引擎”。

实际的分词并不总是如此简单，这种前缀优先的最大长度匹配的方法对于“学历史学好”这样的句子会分成“学历/史学/好”这样的错误结果，那么如何避免呢？

使用后缀树方法可以有效地降低这种错误，不过在从一个字典转到一个后缀树多少有些不够自然。后缀树的结构如图 4-16 所示。

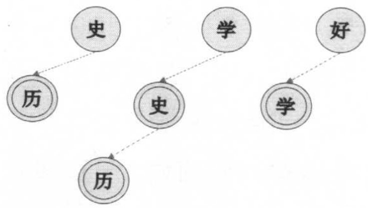


图 4-16 后缀树结构的词典组织形式

与前缀树的构造方法不同，树根表示一个词的结尾字，双圈字表示某个词的词头。我们知道，“历史学”、“历史”和“学好”是一个词，“学历史学好”被切分成“学/历史/学好”。

从句子的右边向左边读，首先找到“好”这个分支。然后读入“学”，恰好存在这样一个词，因此抽取出它。继续读入“史”，依次读入全部句子，直到切分结束，最后切分的结果为“学/历史/学好”。

虽然没有错分出“学历”，但是因为“好”这个字的特殊性，这样的逆向切分仍然不能达到理想的结果。

第 1 种方法称为“最大正向匹配法”（Maximum Matching Method），通常简称为“MM 法”；第 2 种方法称为“逆向最大匹配法”（Reverse Maximum Matching Method），通常简称为“RMM 法”。RMM 法的基本原理与 MM 法相同，不同的是分词的扫描方向。

这类分词的方法可以称为“贪婪算法”，“贪婪”在于总是认为最大匹配的词汇最优。或者说最大匹配可以使得切分出的词汇数量最少，而较少的索引词可以降低索引系统的工作量和最终索

读书笔记

引文件的大小。

而事实上，贪婪并不总是最优。例如对于"学历史学好"，如果正向最大匹配，那么切分为"学历/史学/好"；如果逆向最大匹配，将会切分为"学/历史/学好"。无论如何切分都不理想，因此"贪婪"导致的局部最优很可能不是全局最优的问题被凸现出来。

不论 MM 法，还是 RMM 法都有不足。一个称为"N-Gram"的方法可以满足由于分错词而带来的损失，简单的如 2-Gram 方法，这种切分方法照顾了所有的可能。举个例子，"走进搜索引擎"按照 2-Gram 的切分方法，得到"走进"、"进搜"、"搜索"、"索引"和"引擎"。这有点类似于前面提到的 Shingle 方法，区别是 Shingle 方法由于是子文档集合。因此会在内部对相同的 Shingle 只保留一个 Shingle，而去掉其他相同的，从而保证集合的惟一性。

"N-Gram"的分词方法虽然有效地避免字典分词的错误而导致索引不完整的情况。但是分词是为了在索引阶段尽量减少索引项，而"N-Gram"的方法，特别是"2-gram"的分词法却会带来大量的关键词索引项，因为例如"进搜"这样的词都做成了索引项，这是极不经济的。

综上所述，没有一种分词方法能够解决全部的问题。虽然采用字典分词的方法作为主流的分词方法能够解决大部分的分词问题，但是字典总是滞后于语言的发展。很多新兴词汇需要每隔一段时间才能成批地收录进词典。因此能够及时地、自动地、准确地发现新词才能把字典分词的威力发挥到最大。新词发现主要通过统计推断的方法来实现。

通过统计学方法实现分词

虽然字典解决了分词的大部分问题，但是由于字典收录词数的限制，分词还需要具有新词发现的能力，即补充新发现的普遍被采用的流行的各种词汇到字典中。其中包括演艺明星姓名和网络流行语,甚至股票代码和火车车次都可能成为新词发现的目标。

事实上，发现新出现的词汇（学术上称为“新词发现”）相当复杂。需要很多基于概率及信息学方面的知识，在本书中不展开深入研究。基本的原理就是探索那些经常一同出现的字，总是相互出现的字很有可能构成一个词。为此需要分析和探索大量网页内容，这种分析和探索过程均离线完成，最后还会有一个人工确认的过程。

即使到现在，中文分词技术还在不断地向前发展，还没有哪一个分词算法能够解决一切问题，语言学多样化的重重难关激励着更多的科学家展开更加深入的研究。中文分词的方法也被应用到英语处理，主要是手写体识别中。因为在识别手写体时，单词之间的空格变得比较模糊。中文分词方法可以帮助判别英语单词的边界，因此其研究工作越来越引起广泛的关注。

第五节 PageRank 04

网页搜索的本质是网页信息的聚合，把本来很难聚合在一起的网页通过共同包含的关键词聚合起来。网页被聚合后就自然会产生排序问题。归纳起来就是既不能不排，也不能乱排，需要通过科学有效的方法将“好”的搜索结果按照“好”的程度依次排列。当然“好”是一个相当主观的且难以量化的概念，因此如何评价这个“好”成为搜索质量的关键。在搜索引擎的发展史中，Google 发明的 PageRank 无疑是浓墨重彩的一笔。

PageRank 的来由

Google 公司的两位创始人 L. Page 及 S. Brin 在 1998 年提出了 PageRank 的概念[Page, et al.,1998]。在他们提出这个概念时，一方面，万维网的发展正处于信息大爆炸的时期。当时估计大约有 1 亿 5 千万网页，而且不到一年的时间网页的数目就会翻倍；另一方面，网页质量参差不齐，例如文献[Page, et al.,1998]中提到的网页信息“从 Joe 在哪里吃午饭到信息检索的期刊”无所不包，可以说实际有意义的、有价值的，以及经常被用户检索的网

页规模并没有想像中的那么大，一个基本的方法是通过为网页排名使得重要性高且有价值的网页能够被优先检索。PageRank（网页排名）就是在这样的应用背景下诞生的。

PageRank 的基本想法

人们在万维网上冲浪的过程中，常常从一个起始网页开始（例如新闻门户网站），之后被那些带有链接文字（锚文本）所描述的网页吸引，从而点击并打开另一个网页。并依次往复，直到对冲浪感觉厌倦，而正是由于网页和网页通过链接关系构成的这个网络使得网上冲浪成为可能。然而直到 Google 提出了 PageRank 模型，这种网页间彼此链接关系的价值才被挖掘出来，这种挖掘过程也称为"链接分析"（link analyze）。简单地说，如果网页 A 链接到 B，那么表示网页 A 的编写者（网页工程师或者其他网页创造者）对网页 B 的一种认可。或者说网页 A 为网页 B 投了一票，网页 B 的重要性被网页 A 认可。

直觉上说，如下 3 点可以认为是网页重要性的一种评价。

（1） 认可度越高的网页越重要，即反向链接（backlink）越多的网页越重要。

（2） 反向链接的源网页质量越高，被这些高质量网页的链接指向的网页越重要。

（3） 链接数越少的网页越重要。

举个例子，假定在一次比赛中一个网球选手 A 被另一个网球选手 B 击败。定义一个 A 指向 B 的链接表示这种胜负关系。即 A→B，表示 A 输给 B，或者说 A 认可了 B 的厉害。那么很显然，对照上面得出下面结论。

（1） 如果 B 的反向链接越多，即认可 B 厉害的人越多，因此 B 的排名越高。

（2） 如果输给 B 的选手的排名越高，即认可 B 厉害的人也是厉害的人，那么 B 的排名越高。

（3） 如果 B 输给的选手越少，即 B 认可厉害的人越少，那么 B 的排名越高。

综上所述，赢的次数多、赢得对手质量高且输的少的选手的排名高是很自然的，PageRank 的算法的基本思想也来自于此。

在描述下载系统时曾提到过“随机冲浪”模型[Bernardo A. Huberman, et al，1998]，它是 PageRank 的理论基础，该模型描述网民访问网页的如下行为。

（1） 用户随机选择一个网页作为上网的起始网页。

（2） 看完这个网页后从该网页内所含的超链接随机选择一个页面继续浏览。

（3） 沿着超链接重复浏览，直到对某个主题感到厌倦而重

新随机选择另一个网页浏览。如此反复，直到结束。

PageRank 的计算公式

图 4-17 为文献[Page, et al.,1998]的原图。图中我们看到一个直观并简单的计算方法。以最左上角的网页为例，这个网页被 3 个箭头指向，假定共计得到了 100 分。然后通过两个链接（小横线表示网页内部的链接，方块表示一个网页）平均将这 100 分均分给了它所指向的两个网页。平均的意义在于假定用户是随机冲浪的，也就是点击第 1 个链接和第 2 个链接的可能性相等。这种由于指向关系而得到一定的分数，最后通过计算每个网页得到分数的多寡评价网页重要性的方法，就是 PageRank 的基本设计想法，也称作链接关系分析（link analyze）。

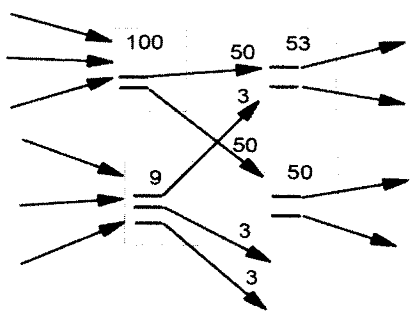


图 4-17　文献[Page, et al.,1998]的原图

如果分数被这样不断地传递下去，因为万维网彼此相连，这样的计算会没完没了吗？或者说 PageRank 的计算会收敛吗？

图 4-18 所示为对三个网页分别计算它们 PageRank 的一个迭

代收敛的例子。

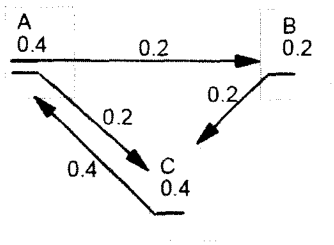


图 4-18　PageRank 的迭代收敛的例子

在图 4-18 种，无论怎样迭代的计算，不难发现，每个网页的得分都是固定不变的。网页 A 的分数平均分给了网页 B 和网页 C，网页 B 又完全给了 C，最后网页 C 把得到的分数又还给了网页 A。可以看出这 3 个网页得到的分数之和总是 1，而且无论如何循环计算，这种分数分布不再发生变化。这种特性在数学上称为“不变分布”，这也是 PageRank 计算收敛的最终奥秘。当然 PageRank 实际的计算公式考虑了各种因素，使得这种不变分布必然存在。这些数学上的原理，请读者参阅相关随机过程的书籍或论文予以透彻了解，下面我们将介绍 PageRank 的计算公式。

PageRank 采用以下公式计算：[Sergey Brin 1998]

$$PR_{\mathrm{n}}(A) = (1-d) + d \times \left(\sum_{i=1}^{m} \frac{PR_{\mathrm{n-1}}(T_{\mathrm{i}})}{C(T_{\mathrm{i}})}\right)$$

说明如下。

（1）$PR_{\mathrm{n}}(A)$：网页 A 的 PageRank 值。

（2） $PR_{n-1}(T_i)$：网页 T_i 存在指向 A 的链接，并且网页 T_i 在上一次迭代时的 PageRank 值。

（3） C（T_i）：网页 T_i 的外链数量。

（4） d：阻尼系数，$0<d<1$，$d\times\frac{PR_{n-1}(T_i)}{C(T_i)}$ 表示在随机冲浪模型中网页 T_i 将自身 d 的份额的 PageRank 值平均分给每个外链。由于网页 T_i 指向网页 A，因此网页 A 获得来自网页 T_i 的 $C(T_i)$ 分之一的 PageRank 值。

从整体上来看这个公式，一个网页 A 的 PageRank 值，由两方面得分决定，其比例分别为 $1-d$ 和 d。

一方面，任何网页都有可能跳转到网页 A 来（不通过链接方式）。假定全集为 n 个网页，那么其中任意一个网页就有 n 个可能的无条件跳转的去处。每种跳转的概率相等，均为 1/n。因此网页 A 得到了每个网页的 $1/n$ 的（$1-d$）的分数，每个网页的分数初始值都为 1。每次分数不论如何流动，其总值都为 n。因此，$\sum_{i=1}^{n}(1-d)PR_n(T_i)/n=1-d$ 的分数被分配到了网页 A 上，PageRank 中 1-d 的分数也可以理解为一个网页的基本分数，这是在整个计算中固定不变的常数。

另一方面，由于有些网页存在明确的指向网页 A 的链接，因此 $d\times(\sum_{i=1}^{m}\frac{PR_{n-1}(T_i)}{C(T_i)})$ 的分数被分配给了网页 A。

因此对于公式：

$$PR_{n}(A)=(1-d)+d\times(\sum_{i=1}^{m}\frac{PR_{n-1}(T_{i})}{C(T_{i})})$$

第 1 个部分可以认为是固有得分；第 2 个部分可以认为是因为被其它网页指向而得到其它网页的一部分得分。

在数学上可以证明，PageRank 由两方面因素共同决定的计算方法可以避免 Dangling page 和 Page sink 的问题[Langville et al 2003]。

PageRank 的计算涉及随机过程方面的数学原理，接下来通过一系列的例子来深入理解。假定存在如图 4-19 所示的简单的网页链接关系。

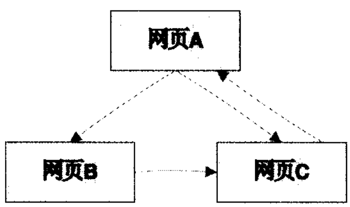


图 4-19　简单的网页链接关系

由 PageRank 的计算方法得到下列方程组，假定 d=0.5：

$PR(A)=0.5+0.5\times PR(C)$

$PR(B)=0.5+0.5\times(PR(A)/2)$

$PR(C)=0.5+0.5\times(PR(A)/2+PR(B))$

解这个三元方程组，得到：

$PR(A)=14/13=1.0769$

$PR(B)=10/13=0.76923$

$PR(C)=15/13=1.1538$

从最后的 PageRank 值可以清楚地看出，在这 3 个网页中，网页 C 的重要性最高，它的得分为 1.1538 分；其次是网页 A；最后是网页 B。这种通过链接关系的分析得到网页重要性是合理的，网页 C 在这个局部是更被“认可”的，它被网页 A 和网页 B 指向；网页 A 被更高级别的网页 C“认可”，而网页 B 被网页 A“认可”，这样网页 A 和网页 B 同样具有一个 Backlink。由于网页 C 的级别大于网页 A，因此最终网页 A 的重要性大于网页 B。

此外，PageRank 还可以通过迭代计算的方法得到，计算过程如表 4-2 所示。

表 4-2　PageRank 迭代计算过程

迭代次数	*PR*（*A*）	*PR*（*B*）	*PR*（*C*）
0	1	1	1
1	1	0.75	1.25
2	1.125	0.75	1.125
3	1.0625	0.78125	1.15625
4	1.07812	0.765625	1.15625
5	1.07812	0.769531	1.15234
6	1.07617	0.769531	1.1543
7	1.07715	0.769043	1.15381
8	1.0769	0.769287	1.15381
9	1.0769	0.769226	1.15387
10	1.07693	0.769226	1.15384

这里初始向量为 $s=[1\quad 1\quad 1]^T$，表示每个网页的初始 PageRank 值均为 1。到达第 7 次迭代时，迭代结果和第 6 次迭代的结果基本相同，因此迭代停止。继续计算下去，PageRank 值不再变化或仅有很小的变化，这种稳定的 PageRank 值被用于衡量网页的重要性。然而当网页数量很大，这种链接关系复杂时，采用这些方法是低效的。

PageRank 的计算方法 ★

在实践中，采用幂法（Power method）来计算 PageRank。

PageRank 的这个经典公式：

$$PR_n(A)=(1-d)+d\times(\sum_{i=1}^{m}\frac{PR_{n-1}(T_i)}{C(T_i)})$$

可以转化为求解 $\lim_{n\to\infty}A^n x$ 的值，其中矩阵为 $A=d\times P+(1-d)\times ee^T/m$（A 也被称为"Google matrix"）。$d$ 为阻尼系数，P 为概率转移矩阵，e^T 为 n 维的全 1 行，m 为全部网页个数（概率转移矩阵大小，x 是各个网页的初始 PageRank 值，初始每个网页的 PageRank 值均为 1。因此，ee^T/m 矩阵在每次迭代时与全部网页的 PageRank 向量乘积总是保持一个 n 维的全 $1-d$ 的向量（在下面的例子中会提到）。这可以理解为总能够从其他网页得到 $1-d$ 的分数，幂法计算过程的伪码如下。

（1） x ← 任意一个初始向量

（2） $r \leftarrow A x$

（3） if（ $\|x-r\|$ ）$<\varepsilon$ ，返回 r

（4） else $x \leftarrow r$, goto 2

使用图 4-19 的例子来理解上面的计算过程。

（1） **P** 概率转移矩阵的计算过程。

初始化一个矩阵 P'，若网页 i 存在一个指向网页 j 的链接，则 $P_{ij}=1$；否则=0，在图 4-19 的例子中，$P'=\begin{bmatrix}0 & 1 & 1\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix}$，然后将每一行除以该行非零数字之和。例如矩阵 P'的第 1 行除以该行数字之和等于 $[0 \quad 1/2 \quad 1/2]$，每一行均按照这样的方法处理得到如下矩阵：

$$P'=\begin{bmatrix}0 & 1/2 & 1/2\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix}$$

这样的矩阵也称为“马尔可夫转移矩阵”（markov transmit matrix），它记录了每个网页跳转到其他网页的概率。对于第 1 行，网页 A 跳转到网页 B 和网页 C 的概率各为 1/2。通常使用其转置矩阵进行计算，即：

$$P'^{T}=\begin{bmatrix}0 & 0 & 1\\ 1/2 & 0 & 0\\ 1/2 & 1 & 0\end{bmatrix}$$

P'^{T} 即

$$A=d\times P+(1-d)\times ee^{T}/m$$

公式中的概率转移矩阵 **P**。

（2） **A** 矩阵计算过程。

由上面的计算结果，$P=\begin{bmatrix}0 & 0 & 1\\ 1/2 & 0 & 0\\ 1/2 & 1 & 0\end{bmatrix}$，且易知 $ee^{T}/m=\begin{bmatrix}1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\end{bmatrix}$，不妨设 d=0.5，这样得到 $A=\begin{bmatrix}1/6 & 1/6 & 2/3\\ 5/12 & 1/6 & 1/6\\ 5/12 & 2/3 & 1/6\end{bmatrix}$，初始每个网页的 PageRank 值均为 1，即 x^{T}=（1,1, 1）。

（3） 循环迭代计算 PageRank 的过程。

第 1 步

$$Ax=\begin{bmatrix}1/6 & 1/6 & 2/3\\ 5/12 & 1/6 & 1/6\\ 5/12 & 2/3 & 1/6\end{bmatrix}*\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}=\begin{bmatrix}1/6+1/6+2/3\\ 5/12+1/6+1/6\\ 5/12+2/3+1/6\end{bmatrix}=\begin{bmatrix}1\\ 0.75\\ 1.25\end{bmatrix}=r,$$

由于 x 与 r 的差别较大，所以 r 赋值给 x，继续计算。

第 2 步 $Ax=\begin{bmatrix}1/6 & 1/6 & 2/3\\ 5/12 & 1/6 & 1/6\\ 5/12 & 2/3 & 1/6\end{bmatrix}*\begin{bmatrix}1\\ 0.75\\ 1.25\end{bmatrix}=\begin{bmatrix}1.125\\ 0.75\\ 1.125\end{bmatrix}$，反复迭代，

最终的计算结果依次为 $\begin{bmatrix}1.062\,5\\ 0.781\,25\\ 1.156\,25\end{bmatrix}$，$\begin{bmatrix}1.078\,125\\ 0.765\,625\\ 1.156\,25\end{bmatrix}$，$\begin{bmatrix}1.078125\\ 0.769531\\ 1.152344\end{bmatrix}$，…，$\begin{bmatrix}1.076\,9\\ 0.769\,23\\ 1.153\,8\end{bmatrix}$ 和 $\begin{bmatrix}1.076\,9\\ 0.769\,23\\ 1.153\,8\end{bmatrix}$。

在最后两次的结果近似或相同的情况下迭代结束，这个结果和解方程、简单迭代的方法求出的结果是一致的。用幂法计算 PageRank 总是收敛的，即总会在有限的迭代计算轮次内结束。

参考文献[Langville et al 2003] [Haveliwala et al 2003] [Lars Elden 2003]可以进一步深入理解 PageRank 计算过程收敛的奥秘，以及参数 d 是如何影响收敛速度的基本原理。

在实际应用中，计算 PageRank 的过程要比想像的更为复杂。例如从提高收敛效率角度，加快计算速度；采用将 Google matrix（PageRank 这个概率转移矩阵学术上也称作 Google matrix）分块的思想执行并行计算减少磁盘 I/O。从而发挥多 CPU 的作用，有兴趣的读者可以进一步探索。

自从 PageRank 的计算方法被公开后，各种为了提高排名而进行的作弊行为也随之而来。Google 及其他采用类似技术的搜索

引擎公司都面临着前所未有的压力，PageRank 的科学性和可信度也受到了极大怀疑。但是科学家们采用了各种反作弊的技术手段，沉重打击了作弊者的不道德行为，维护了 PageRank 应有的价值。

至此，我们结束了分析系统的全部旅程，为了更好的从全局理解分析系统各个模块的工作关系，下一节中将和大家一起从宏观全局的视角重新回顾分析系统

04 第六节 分析系统结构图

分析系统在搜索引擎的架构中承担了网页结构化、网页消重、文本分词及 PageRank 计算 4 项基本任务。通过前面的分块学习，最后通过一个分析系统结构图来全面了解分析系统的运作方式。分析系统结构图如图 4-20 所示。

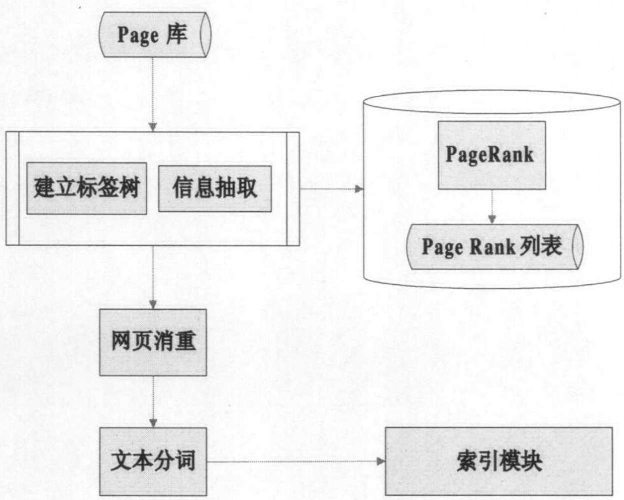


图 4-20 分析系统结构图

其中 Page 库是下载系统通过爬虫下载到的原始网页，分析系

统通过以下步骤完成对这些网页的分析工作。

（1）经过一个网页结构化的过程，包括建立标签树并从网页中抽取有价值的属性，完成从原始网页打包成一个网页对象的过程。

（2）网页消重模块丢弃冗余的页面，相似或相同的网页仅保留一个传给分词模块。

（3）文本分词将正文切分成以词汇为单位的集合。

（4）将分析的结果发往索引模块，进行索引入库。

以上 4 个步骤中网页结构化、消重、分词这三项工作同步计算，因此速度非常快。仅 PageRank 的计算非常耗时，而且必须积累一定数据后才能生成一次数据（这里信息抽取过程中得到的网页链接信息发往 PageRank 计算服务器）。由于执行一次 PageRank 的计算代价极大，因此采用离线计算方法。离线计算的结果是一个 PageRank 列表，其中包含每个网页的一个 PageRank 值。该值越高，网页的重要性越高，在检索时就越容易被检索到。在索引系统中会利用该 PageRank 值对文档列表索引项的先后顺序产生影响，也就是越是重要的网页（PageRank 值高）在索引中越能够占据有利的位置。

回顾本章，网页分析系统的工作层层深入。除了第 1 步分析工作是为本层服务，其余的分析工作包括网页消重、分词及 PageRank 的计算都是为了索引系统查询系统服务，因此只有在深

入了解索引系统和查询系统的原理后我们才能深刻理解这些分析系统工作的重大意义。接下来我们将继续走进搜索引擎的索引系统中，领略索引系统的魅力。

参考文献

[1] [Abdur chowdhury et al 2002]Collection Statistics for Fast Duplicate Document Detection ABDUR CHOWDHURY, OPHIR FRIEDER, DAVID GROSSMAN, and MARY CATHERINE McCABE Illinois Institute of Technology ACM Transactions on Information Systems, Vol. 20, No. 2, April 2002, Pages 171–191

[2] [Bernardo A. Huberman, et al，1998] Bernardo A H,Peter L T P, James E, et al. Strong regularities in world wide web surfing. Science,1998

[3] [Broder 1997]A. Broder, "On the Resemblance and Containment of Documents," *sequences*, p. 21-29, Compression and Complexity of Sequences 1997, 1997

[4] [C.-N.Hsu, et al.,1998] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. Information Systems, 23（8）:521-538，1998

[5] [Haveliwala et al 2003] Haveliwala, T.H. and Kamvar, S.D.. The second eigenvalue of google matrix. Stanford University Technical Report , 2003

[6] [Hammer,et al.,1997] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the web. In Proceedings ofthe Workshop on Management of Semistructured Data, pages 18-25 , May 1997

[7] [J. Hammer et al.,1997]J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo Extracting Semistructured Information from the Web

[8] [Lars Elden 2003] Lars Elden, The Eigenvalues of the Google Matrix Technical Report LiTH-MAT-R-04-01

[9] [Langville et al 2003] Langville, A.N. and Meyer, C.D. Deeper inside pagerank. Technical Report, NCSU Center for RES SCI Comp. 2003

[10] [Page, et al.,1998] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford Digital Library Technologies Project 1998

[11] [S.-H.Lin et al,.2002] S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. SIGKDD , 2002

[12] [S. Brin 1998] S. Brin and L. Page. The anatomy of a

large-scale hyper textual Web search engine. Computer Networks and ISDN Systems, 30 (1-7) :107-117 , 1998

[13] [Sergey Brin et al 1995]Sergey Brin , James Davis , Héctor García-Molina, Copy detection mechanisms for digital documents, Proceedings of the 1995 ACM SIGMOD international conference on Management of data, p.398-409, May 22-25, 1995, San Jose, California, United States

[14] [梁南元，1987] 梁南元，"书面汉语自动分词系统——CDWS" 中文信息学报 1987 年 第 02 期

[15] [李晓明 2004] 李晓明，闫宏飞，王继民，"搜索引擎——原理、技术与系统"，科学出版社

第五章

搜索引擎的索引系统

05 第一节 知识准备

在搜索引擎的 4 大系统中，第 3 个系统称为“索引系统”。该系统就好像搜索引擎的数据大本营，在这里存储了并索引了数以亿计的网页。在搜索引擎早期的发展中，能够索引的网页数量代表了整个行业的技术发展水平。由于需要支持多用户同时检索，索引系统还必须提供低于秒级的检索时间，因此“存得下”和“查得快”是围绕本节的重要话题。

与前几个章节一样，一起来做一下热身活动，了解一些基本概念。

信息

信息是能够被传达和理解的消息，是通过学习和经历获得的知识，是用来做出判断的一组事实[WordNet]，不同的角度上对信息具有不同的解释。这里我们认为信息就是结构化的网页数据，即一组有价值的数据的集合。

索引

索引也是一种信息，可以说是信息的信息，或者说是描述信息的信息。例如，书中包含的目录。其中每一条目就是一个索引，用来标识某个章节的页码。帮助读者快速浏览，索引就是这样一种短小精炼的检索信息的信息。

倒排索引、倒排表、临时倒排文件、最终倒排文件

为了便于理解整个章节笔者做出如下定义：

倒排表是指存放在内存中的能够追加倒排记录的倒排索引。倒排表是迷你的倒排索引。

临时倒排文件是指存放在磁盘中，以文件的形式存储的不能够追加倒排记录的倒排索引。临时倒排文件是中等规模的倒排索引。

最终倒排文件是指由存放在磁盘中，以文件的形式存储的临时倒排文件归并得到的倒排索引。最终倒排文件是较大规模的倒排索引。

倒排索引作为抽象概念，而倒排表、临时倒排文件、最终倒

排文件是倒排索引的三种不同的表现形式。

其他概念

索引部分概念很多，因此本章第二节~第四节分别介绍全文检索、文档编号、正排索引、倒排索引的基本概念。在集中理解索引系统的主要概念后，接下来再了解索引创建中的一些计算细节。

第二节 全文检索 05

全文检索（full-text retrieval）技术的出现是信息检索领域的一场革命，它细化了信息检索的粒度。提供了实现多角度，多侧面且全新的信息检索体验。因此搜索引擎全面采用了这种崭新的技术，并使之成为主流的检索方法。

全文检索

早期的信息检索主要通过检索数据信息的外部特征，例如标题、作者、摘要、附录及资料的编号等。这样的检索系统常见于图书馆的馆藏图书检索中，它主要存在如下两个大问题。

（1）检索结果排序不理想。

（2）只能对标题进行检索。

出现这些问题是因为没有考虑到文档内容（本章使用文档笼统地代表书籍或者网页）。全文检索顾名思义，是对文档的全部信

息进行检索，这些信息包括标题和正文等。简单地说，全文检索的内在本质归纳起来就是如下两条。

（1）文档的全部文字参与索引。

（2）检索结果能够提供检索词出现的实际位置。

在全文检索的过程中，只需要用户提供一个或多个检索关键词（以下简称“关键词”)。不仅能够检索出命中的文本，还能够提供这些关键词在文本中出现的位置。受到搜索引擎检索结果的展示窗口的限制，不可能把全部关键词出现的位置一一列出，最终的展示效果如图 5-1 所示。

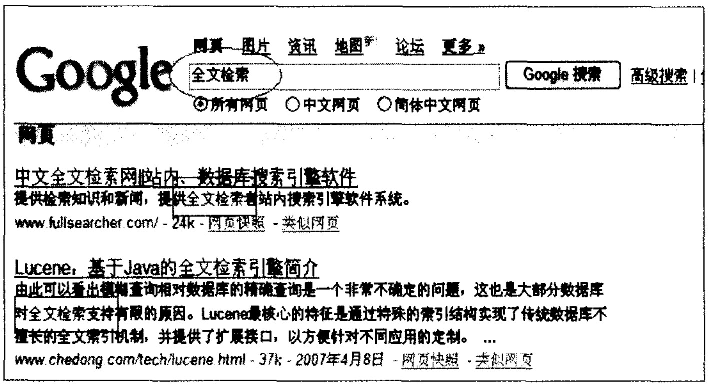


图 5-1　关键词位置的展示方式

其中椭圆矩形表示网民向搜索引擎提交的关键字，矩形框表示关键字在文档中的位置。这里搜索引擎采用一种称为“标红”的技术用红色标出关键词，其余文字为黑体，这样的方法比用数字标出位置信息更加直观。接下来用户就可以通过上下文、关键词在文本中的位置，以及检索到的文档标题等因素确定哪一条才

是符合需要的检索结果。例如图 5-1 是用“全文检索”这个关键字在 Google 搜索引擎中进行检索的结果。

当今搜索引擎无一例外地采用了全文检索，使得越来越多的人开始使用搜索引擎查找信息，信息检索的面貌也为之一新。在这革命性的搜索体验背后蕴涵的是大量的信息组织的智慧，它最终使得全文检索从梦想成为现实。

05 第三节 文档编号

一个惟一的编号是描述一个复杂事物最好的方法，每一个文档赋予一个惟一的编号后，这个编号就能代表这个文档。

编号的本质

编号的历史能够追溯到古老的时代。部队有番号、单位职工有工作证号、大中专院校学生有学号，以及公民有身份证号。从各种各样的编号中不难发现这样的规律，即凡是需要区别大量个体的情况就会用到编号。一个家庭成员之间不会用编号来区分，因为用姓名更加自然并亲切。然而，如果某个家庭成员来到学校读书，因为可能存在同名同姓的同学。因此采用姓名就不能有效地区分，使用学号就成为必然的手段。这样通过一定的编码方式可以保证某个年级某个院系某个班级的某个学生拥有惟一标识自身的学号。

对于搜索引擎来说，编号不仅仅是保证惟一，而且它好像是一道连通信息和索引的桥梁。搜索引擎需要处理的信息是海量的，

为了能够快速检索到这些信息，通常的做法是为信息创建索引。索引也是信息，是那种能够用来找到信息的信息。对于搜索引擎来说，网页是信息，关键词就是索引，索引相对于信息来说是轻量级的。文档千差万别，而关键词总是有限的。因此虽然处理的是海量信息，但是转化到索引后在空间占用上就会变得相当小。检索在索引上完成后，得到一组匹配关键词的文档编号。继而将这些文档编号在文档信息库中取出，通过一系列计算展示出来，编号就是这样在检索中发挥了重要的作用。

一个网页被爬虫抓取，经过分析系统的分析。然后得到结构化的网页对象，为此首先需要为结构化的网页对象进行编号存档。在索引系统中，我们统一将这种结构化的网页对象（包含标题、正文和 URL 等信息的结构体）称为“文档”，网页的编号称为“文档编号”。

文档编号和日常生活中的编号最大的不同在于，它不需要具有具体的含义。与机器不同，日常生活中使用编号大多具有某种意义。例如我们的身份证号每一段都有其特定的含义，然而文档编号有其特殊性。在信息索引阶段，编号无处不在。各种检索及排序计算都依赖于编号，因此文档的编号方法会有一些额外的特别要求。

文档编号的方法

文档编号需要满足下面 3 个条件。

（1）任何一个文档在其生命周期中仅有一个编号。

（2） 任何两个不同的文档的编号不同。

（3） 编号在计算中尽可能高效，并且为了便于存储，要越短越好。

第 1 个条件表明文档编号和文档内容无关，有些文档内容常常发生变化。如果编号与内容有关，编号的稳定性将不能够被满足。

编号不能直接由文档文件名代替，因为很多网站都有类似 index.html 和 welcome.html 等这样通用的文件名。如果编号由文件名代替，那么就破坏了第 2 个条件，即不同的文档由于相同的文件名而赋予相同的编号。

综上所述，文档内容和文件名均不能代表一个文档。事实上，只有 URL 才能够惟一地表示一个文档。通过浏览器输入一个 URL 只能到达惟一网页，这首先保证了第 2 个条件；其次也满足了第 1 个条件，网页的内容变化不会影响其 URL。因此，URL 是天然的计算文档编号的来源。

由于中文万维网网页规模将来必然达到百亿量级的规模，如何合理地对文档进行编号才能符合第 3 个条件要求的高效计算呢？

显然将文档的 URL 直接作为编号是不合适的，一个 ASCII 字符为一个字节，这样的一个 URL（http://sports.sohu.com/20070411/n249364 259.shtml）就需要多达 48 个字节。不论是计

算还是存储都是不经济的。计算不经济主要体现在字符串运算慢，这是因为字符串编号在查找和排序时的代价都很大。排序和查找都是利用多次比较完成的，时间的耗费主要在比较操作上，字符串之间的比较耗时远大于整型的比较；存储不经济主要体现在字符串的空间占用较大，增大了磁盘 I/O 的次数。此外，空间占用量大还会影响计算效率。空间占用越大，内存工作集也就相应变大，内存和对换区的换入换出机会就增大。

综上所述，采用对字符串映射为占 64 比特大小的长整形整数，利用整数计算高效，存储代价小的特点进行文档编号是必然的选择。

整型编号不仅可以节约大量存储空间，而且大大提高检索计算中比较排序的效率。采用 URL 字符串签名的方法（参见第三章），一个 32 位整型能够表达 40 亿量级的数。对一个 URL 字符串签名得到一个占两个整型（8 个字节）大小的 MD5 签名，理论上它能够表达足够多的网页数量，这被认为符合当前时代万维网网页数量的编号要求。对于大规模的网页编号来说，两个整形的空间耗费还是太大。通过编码的技巧可以让其变得更短，更适合计算的需要。

游程编码

本节介绍一种通用的编码方式，称为“游程编码”，使用这种方式进行编号长度压缩。在后面还会提到这个技巧，因此这里可以作为一个知识准备。

假定一个文档编号序列为 1，17，34，69，489，512，3456，这些都是文档的内部编号。编号为升序序列，即排在后面的数不小于排在前面的数。不妨假定编号为一个整型大小，并且能够满足 40 亿个不同个体的编号需要。上面这个文档编号序列包含 7 个整数，因此需要 7 个整型（28 个字节）的数组来存储。

对于这种单调的序列，可以将增量整数序列被变换为差分序列，这种编码方式也称为“差分编码”（gap encoding）。通过转换得到 1，17-1，34 -17，69-34，489-69，512-489，3456-512，即 1，16，17，35，420，23，2944。由于差分序列除第 1 个编号外其余都保存的是间隔距离，因此带来一个好处和一个坏处。

好处是因为存放的是相对数，所以除第 1 个数以外，序列中的数都变得较小。

坏处是为了取出序列中某个特定值，需要由头至尾取出所有数据，并进行多次加法运算。一旦缺失一块数据，则无法序列中的后续部分无法恢复。

举个例子，在 1，16，17，35，420，23，2944 这个序列中，如果需要取序列中第 7 个数，首先必须读出全部数据，进行连加，也就是需要计算 1 + 16 + 17 + 35 + 420 + 23 + 2944 = 3456。如果由于各种原因，中间的数据缺失或者出现异常，例如第 4 位应该存放 35 而读取出来为-35，这使得后续的数据无法恢复。通过这个例子不难看出，一方面付出了 CPU 的代价；一方面付出了维护数据完整性的保证，继续来看得到怎样的补偿。

由于序列中的数都变得很小，所以可以采用一种变长编码

（Variable Byte Coding）[Williams 1999]的技巧。这是一种字节对齐的编码方式，即将整数转化成二进制后，以 7 位为单位对其分段。每段段尾加一位成为 8 位，恰好用一个字节表示。末位为 0 表示该段是最后一段，为 1 表示还有后续段。例如，135 的二进制编码表示为 10000111 通过编码后为 00000011 00001110。第 1 段（00000011）末尾的 1 表示该字节为段首（还有后续段），倒数第 1 个 1 表示 135 二进制数据中的第 1 个 1。第 2 段（00001110）末尾的 0 表示该段不是段首，其余 7 位表示 135 二进制数据中的后 7 位（0000111）。编码的过程很容易，解码的过程也相当快。

在实践中，Variable Byte Coding 编码不是压缩率最高的编码方式，例如其压缩率不如 γ code（Elias），但由于 Variable Byte Coding 具有字节对齐的优秀特性，所以解码非常快速。具体的解码过程参考下述伪码。

解码过程的伪码为：

```
VariableByteRead（A）
int i = 0x1;
int v = 0;
while（（i & 0x1） == 0x1 ）
    i←HEAD（A）;
    v←（v<<7） +（（i>>1）&0x7F）;
return v;
```

在上述伪码中，HEAD（A）表示从 A 中读取头部的一个字节，并去掉这个字节。例如，135 表示为 00000011 00001110。

第 1 次执行 HEAD 操作返回 A 的段首字节，该字节的二进制表示为 00000011 并且去掉这个头部字节。因此此时 A 的二进制表示为 00001110，这时 i 保存段首字节（00000011）。对其右移一位（i>>1），取其字节后 7 位（&0x7F），此时 i 为 00000001。然后 v 初始化为 0 ，这样第 1 次执行后 v=1。

第 2 次执行 HEAD 时，返回的结果是字节（00001110），并存放到变量 i 中，即十进制的 7。将变量 i 右移动 1 位，并取后 7 位，得到 00000111，加上第 1 次计算的 v 右移 7 位，最后得到 10000111，即 135。

对于 32 位的整数，采用 Variable Byte Coding 编码方式。小于 128 的数可用 1 个字节来表示，从而节省了 3/4 的空间；而大于 268 435 455（二进制 1111111-1111111-1111111-1111111 共 24 位，每位存储 7 位，因此需要 4 个字节存储 268 435 455）的数则需要 5 个字节。这样大于 268 435 455 的数用 Variable Byte Coding 编码方式需要的存储空间超过了一个整形的大小，不但没有达到压缩的目的，反而增大了空间。但是由于经过排序，序列之间的差值都会保持相对较小，因此差值达到如此大的距离（大于 268 435 455）的概率极小。可以忽略这种情况带来的代价，整体的存储是经济的。此外，由于计算机以字节为操作单位，字节对齐的编码往往更能发挥硬件的优势，所以可变字节编码的压缩和解压缩的速度都较快。

这个差分序列中的 1，16，17，35，23 都小于 128，因此只需要 1 个字节。420 需要两个字节（420 的二进制表示为 110100100，每个字节只有 7 个有效位，因此这里需要两个字节）。同理，2 944 也需要两个字节。这样全部序列需要 9 个字节，和

原序列需要 28 字节相比压缩了 60%。由于这些表示文档编号的序列存放在磁盘中，因此节约存储相当于减少了磁盘 I/O 的次数。压缩存储是典型的以时间换空间，一方面由于多了编码和解码的过程，导致 CPU 的消耗增加；另一方面由于节约了磁盘，而减少了磁盘读取次数。一般来说，I/O 是索引系统的瓶颈。而且 CPU 和 I/O 之间性能差距还在不断扩大，因此采用这种技巧带来的减少了磁盘 I/O 的次数的好处完全可以抵偿因此带来的 CPU 计算量增大的不利结果。

文献[Witten 1994]堪称是信息检索的经典书籍，其中介绍大量关于大规模数据压缩的其他方法和技巧，它们各有优劣。需要进一步钻研的读者可以通过阅读本书全面地理解处理大规模数据的各种技巧，通过 Variable Byte Coding 的学习，充分理解排序对压缩的关键作用，排序带来的一些冗余数据和处理这种冗余数据的方法。

在解决了文档编号存档的问题后，接下来就是如何查找到这些文档。记住编号超出人类记忆的能力范围，记住标题也难度颇大。那么对于酒香还怕巷子深的问题，用什么方式来索引这些文档才能符合全文检索的需求呢？下一节我们将着重介绍有关倒排索引的概念。

05 第四节 倒排索引

全文检索的检索是通过关键词来进行检索，因此为关键词建立索引是很自然的。我们把按关键词创建的索引称为“倒排索引”，在这里关键词称为“索引词”，因为并不是所有的关键词都会创建索引。

经典的倒排索引

笼统的说，索引系统的索引如图 5-2 所示。

索引系统中的索引[S. Brin 1998]包含了 3 个概念，分别是命中(Hit)、正排索引(Forward Index)和倒排索引(Inverted Index)。接下来就围绕这三个概念展开，首先理解这 3 个概念。Hit 表示索引词在文章中的位置 (position) 和字体等信息，这里为了突出重点，忽略 Hit 的影响，集中精力了解正排索引和倒排索引。

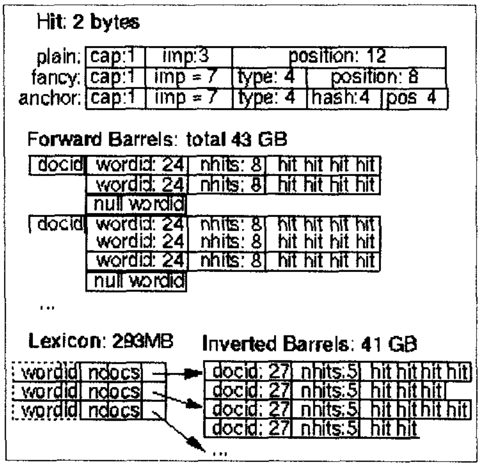


图 5-2　索引系统的索引

正排索引（前向索引）

正排索引也称为“前向索引”。它是创建倒排索引的基础，具有以下字段。

（1） LocalId 字段（表中简称 “Lid”）：表示一个文档的局部编号。

（2） WordId 字段：表示文档分词后的编号，也可称为“索引词编号”。

（3） NHits 字段：表示某个索引词在文档中出现的次数。

（4） HitList 变长字段：表示某个索引词在文档中出现的位

置，即相对于正文的偏移量。

由于一篇文章中的某些词可能出现多次，而且位置不同，而全文检索的本质要求是把这些位置标识出来，因此 HitList 中的每个命中都表示索引词在文档的某个位置中出现了一次，这个序列为单调递增序列。基于游程编码的方法，变升序序列为差分序列，采用前文提到的 Variable Byte Coding 方法编码可以大大压缩正排索引的 HitList 字段。

事实上，文档编号在不同的计算过程里，分别称为局部编号和全局编号，它们在编号长度上稍有不同，为了便于理解，本书不区别这些区别，认为 LocalId 和 DocId 完全一致，都表示一个文档的唯一编号。在正排索引中 LocalId 采用升序序列编号（假定编号采用自增 1 的方式递增），这为下面的计算创造条件。进行倒排索引的转化时，由于正排索引中 Lid 天然的有序性，因此在正排索引转化为倒排索引的创建过程中，自然可以保证倒排索引中每个词汇对应的文档编号也是有序的，倒排索引将在下一节中介绍。

这样，正排索引如图 5-3 所示。

LId	WordId	NHits	HitList
Doc1	Word1	m	Hit1,...Hitm
	Word2	n	Hit1,...Hitn
	...		
	WordN		
	NULL		
Doc2	...	...	...
	NULL		

图 5-3　正排索引

通过一个例子来了解正排索引的创建过程。假定存在这样一个编号为 1 的文档，其全文为“走进搜索引擎，学习搜索引擎”，分词的结果为“走进/搜索引擎/学习/搜索引擎”。不妨为“走进”编号为“T1”，“搜索引擎”编号为“T2”，“学习”编号为“T3”。通过计算得到“走进”出现 1 次，出现位置为 1；“搜索引擎”出现两次，出现位置为 3 和 9（图中存放的为未压缩的差分序列 3 和 6）；“学习”出现 1 次，出现位置为 7。创建的正排索引如图 5-4 所示。

LId	WordId	NHits	HitList
1	T1	1	1
	T2	2	3,6
	T3	1	7
	NULL		

图 5-4　正排索引示例

HitList 是变长的，因此需要 NHits 这个字段标记其长度，这样才能读出全部的正排索引数据。假定每个域都是一个字节大小，而 HitList 变长。在上面这个例子中，假定这个正排索引依次存放在一个数组 Array 中，则当读取到 Array[2]时，读取的内容为 T1 的 NHits，读出的结果为 1。由于 T1 出现了 1 次，因此在 T1 的 HitList 中仅存放了一个位置信息。这样在读取 Array[3]后，接下来读取 Array[4]则能够判断为下一个索引词 T2。正是由于 NHits 的这种表示长度的作用，全部的数据才能被有效地读取。

最后用 NULL 表示为一个结束符的这种设计是很巧妙的；否则，正排索引可能是这样，如图 5-5 所示。

读书笔记

LId	WordId	NHits	HitList
1	T1	1	1
1	T2	2	3,6
1	T3	1	7

图 5-5　冗余存放 DocId 的正排索引

在图 5-5 中，为了结构化数据的需要，在去掉表示结束符的 NULL 后，正排索引必须冗余地存放 DocId，因此一个标记结束符的字段有效地压缩了正排索引的大小。

本质上说，正排索引以文档编号为视角看待索引词，也就是通过文档编号去找索引词。任给一个文档编号，能够知道它包含了哪些索引词、这些索引词分别出现的次数，以及索引词出现的位置。然而全文索引是通过关键词来检索，而不是通过文档编号来检索，因此正排索引不能满足全文检索的要求。

虽然正排索引不能满足全文检索的需要，但是正排索引为创建倒排索引创造了有利条件,是计算倒排索引的不可缺少的一环。

倒排索引

倒排索引是一种以关键字和文档编号结合，并以关键字作为主键的索引结构。倒排索引分为两个部分。

（1） 第 1 个部分：由不同索引词（index term）组成的索引表，称为“词典”（lexicon）。其中保存了各种中文词汇，以及这些词汇的一些统计信息（例如出现频率 nDocs），这些统计信息用于各种排名算法（Ranking Algorithm） [Salton 1989;Witten 1994]

读书笔记

（2） 第 2 个部分：由每个索引词出现过的文档集合，以及命中位置等信息构成，也称为“记录表”（posting file）或“记录列表”（posting list）。如图 5-6 所示。

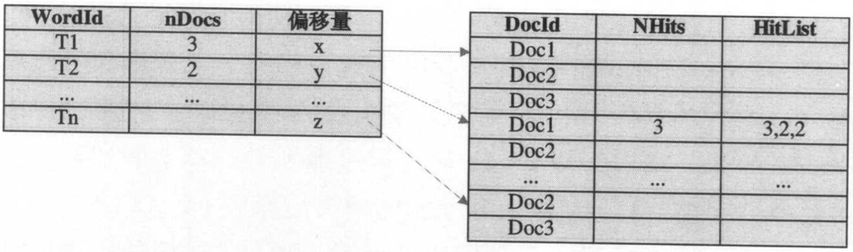


图 5-6　倒排索引

左边的表结构（词典）记录索引词 Id 号、匹配该索引词的文档数量，并匹配文档在记录文件内的偏移量，通过这个偏移量就可以读取记录文件对应区域的信息。例如在图 5-6 中，通过读取 T1 的偏移量 x，读取在记录文件中 T1 命中的相关文档 Doc1、Doc2 和 Doc3 的相关信息。

右边的表结构（记录表）记录文档编号（DocId）、索引词在该文档的命中个数（NHits），以及命中域的列表（HitList）。例如在图 5-6 中，记录文件显示了 T2 关键词在 Doc1 中出现了 3 次。

在如图 5-6 所示的倒排索引示例中，以索引词 T2 为例，T2 在两个文档中出现。通过偏移量在记录文件中找到了存放与 T2 有关的信息，即 T2 匹配的 Doc1 和 Doc2 两个文档，并且在 Doc1 中出现了 3 次，位置分别为 3、5 和 7（图中 3、2 和 2 表示为差分序列后的实际值，即 3、5-3 和 7-5）。

注意到这种按照索引词组织文档的方式，在索引词 WordId

读书笔记

一边，其 Id 号不会重复；而在 DocId 一边，由于每个文档都可能包含多个索引词，DocId 的重复非常普遍，因此对 DocId 就需要进行大规模的压缩。

压缩编码也采用 Variable Byte Coding 编码方法进行压缩，首先对 DocId 排序，接下来将 DocId 的递增序列为差分序列，最后用 Variable Byte Coding 编码方法进行压缩编码。例如这样的 DocId 序列（22，5，9，1），通过排序得到（1，5，9，22），变差分序列得到（1，4，4，13），压缩编码后得到（2，8，8，26），对于小于 128 的数进行压缩编码，相当于乘 2（详细参见本章第三节中的相关内容）。

最后，在倒排索引中，按照何种顺序存放 DocId 更加有利于检索呢？在图 5-6 中，T1 这个词有 Doc1、Doc2、Doc3 与之相匹配，也就是在这些文档中都出现了 T1，那么在倒排索引的记录表中，哪个文档编号先存放，哪个文档编号后存放呢？这种存放顺序的策略大致有如下 3 种。

（1） 按照 DocId 升序存放。

（2） 按照索引词在文档中出现次数降序存放。

（3） 记录表分块存放,块内按 DocId 升序存放，块间按 PageRank 值降序存放。

对于方案 1 来说，它有助于在多关键词查询中，得到相同的 DocId（在第六章中介绍查询系统时，会提到因为 DocId 有序而为

查询带来的好处）。并且能够对 DocId 进行压缩编码，同时降低磁盘 I/O 的开销。

对于方案 2 来说，按照索引词 Id 在文档中出现的次数降序排序，因为索引词出现次数多的文档与查询关键词的相关性越好（这里，索引词和关键词是同一个词，在索引系统中称为“索引词”，在查询系统中称为“关键词”或“查询词”），这是很自然的，检索就是检索哪些与查询词相关性高的文档。文献[S. Brin 1998]阐述了一种折中的设计方案，即不同的索引词命中区分对待，对于索引词在标题或者锚文本（Anchor）命中的文档，以及命中信息存放在一个记录表中，不妨称为“表 A”，表内数据按 DocId 排序；命中其他位置的文档及其命中信息存放在另一个记录表中，称为“表 B”，同样表内数据也是按照 DocId 排序。这样对于每个索引词的查询，优先在表 A 中查询。只有当结果数不够时，再到表 B 中查询。这样既照顾到了压缩存储的需要，也照顾到了相关性，通常标题中的词大多在正文中会多次出现。当然这种折中的方法有时也不够合理，它过分倚重于锚文本的关键词，常常被针对这种算法作弊的网页设计者利用。

对于方案 3 来说，一方面照顾到了方案 1 的索引压缩功能（文档按序存储）；另一方面照顾了重要的文档在检索过程中优先被检索的需要，而且防止了方案 2 中由网页设计者作弊可能带来的麻烦。有些网页设计者在网页中正文，锚文本中堆砌大量经常被查询的重要关键词，因此方案 2 在设计上不可避免的缺陷容易被利用。当然 PageRank 算法也会被作弊者利用，然而 PageRank 的作弊较为困难，所以方案 3 是较为理想的解决方案。

最后，总结一下正排索引和倒排索引的关系。本质上说，存

在这样两个空间，一个称为"索引词空间"，一个称为"文档空间"。正排索引可以理解成一个定义在文档空间到索引词组空间的一个映射，任意一个文档对应惟一的一组索引词；而倒排索引可以理解成一个定义在索引词空间到文档组空间的一个映射。任意一个索引词对应惟一的一组该索引词其命中的文档。因此从文档到正排索引，进而从正排索引到倒排索引就是理顺这种关系的过程。使得给出一个索引词，就能通过倒排索引能够找到其命中的文档，以及位置信息。

第五节 数据规模的估计 05

在介绍索引系统主要概念后，通过索引数据规模估计的计算方法来体验索引系统设计中必须考虑的数据规模问题，我们首先从齐普夫法则（zipf law）开始说起。

齐普夫法则

齐普夫于 1902 年 1 月出生于一个德裔家庭（其祖父在 19 世纪中叶移居美国），1924 年以优异成绩毕业于美国哈佛学院。1925 年，齐普夫在德国波恩及柏林学习比较语文学专业。但是以其名字命名的定律却早已走出语言学，进入了信息学、计算机科学、经济学和社会学等众多研究领域，在学术界享有极高的声誉。

简单说来，齐普夫法则可以描述为第 k 个最经常出现的词，其词频（发生率）与 $1/k$ 成正比。

齐普夫是这样得到这个定律的，首先选用一个有关 James Joyce Ulysses（不妨理解为一本中等篇幅的书籍）的用词数据，统计不同词汇的出现次数，并按照出现次数排序。他发现了一个惊人的现象，即一个词汇在词频上的排位（rank）乘以其出现的

次数惊人地接近于一个常数，而这个常数即书籍的实际总词数 260 430，如图 5-7 所示。

I Rank (r)	II Frequency (f)	III Product of I and II (r×f = C)	IV Theoretical Length of Ulysses (C×10)
10	2 653	26 530	265 300
20	1 311	26 220	262 200
30	926	27 780	277 800
40	717	28 680	286 800
50	556	26 500	278 000
100	265	26 500	265 000
200	133	26 600	266 000
300	84	25 200	252 000
400	62	24 800	248 000
500	50	25 000	250 000
1 000	26	26 000	260 000
2 000	12	24 000	240 000
3 000	8	24 000	240 000
4 000	6	24 000	240 000
5 000	5	25 000	250 000
10 000	2	20 000	200 000
20 000	1	20 000	200 000
29 899	1	29 899	298 990

(Zipf 1949 :24)

图 5-7　齐普夫的统计结果

词频排名第 10 的词汇，在书中出现了 2 653 次，排名和出现次数的乘积为 26 530。排名 20 的词汇，在书中出现了 1 311 次。排名和出现次数的乘积为 26 220，可以看出它们的乘积是十分接近的。如果将其看做相等，得到以下公式：

$$occurence(T) * rank(T) = C$$

其中 *occurrence*(*T*)表示词汇 T 的出现次数，*rank*(T)表示词汇在全部出现的词汇中词频排序，C 为一个常数。上述公式同除以实际总词数（*N*），则有：

$$\frac{occurence(T)}{N}=\frac{C}{rank(T)}$$

公式左边表示词汇 T 的词频，这样就得到了一开始提到的齐普夫定律。即第 k 个经常出现的词，其词频与 $1/k$ 成比例，Zipf 在[zipf 1949]中提出这是由于语言上的省力原则起作用的结果。有兴趣的读者可以进一步阅读文献[姜望琪 2005]了解省力原则的社会学原理。

布尔检索模型下的索引规模估计★

通过学习齐普夫法则，接下来利用这个法则粗略地估计布尔检索模型下的索引规模，从实践的角度体验一下索引的规模[Stanford IR]。

首先，由齐普夫法则，第 i 个最经常使用词汇的频率和 $1/i$ 成比率，因此第 i 个最经常用词汇的频率为 c/i。

接着，假定全部汉语词汇数目为 50 万，因此有 $\sum_{i=1}^{500000} c/i=1$。这不难理解，在全世界所有中文书籍中取出 50 万个互不相同的词汇，并统计全部中文书籍的实际总词数。然后分别计算这 50 万的词汇各自出现的数目，用数目除以总词数即得到词频。这 50 万词汇的词频总和可以看做分母（总词数）不变，分子（各自出现的次数）相加。分子之和恰好为总词数，因此除的结果为 1。

在数学上，称 $H_k=\sum_{i=1}^{k}1/i$ 为"调和级数"，因此 $\sum_{i=1}^{500000} c/i=1$

可以简化为：

$$c \times H_{500000} = 1$$

调和级数的计算公式为：

$$H_{500000} \approx \ln 500000 \approx 13$$

这个常数 c 通过计算得到 $c=1/13$。

综上，一个词频排序为 i 的词汇，它的实际词频大约是 1/13i。这里的词频排序可以认为是海量数据中的排序，而不是局限于一本书。对于搜索引擎来说，一个词汇的词频就是它在全部网页中出现的实际次数除以全部网页实际出现的词汇数。例如一个词汇只在一个网页中出现，且出现了 K 次。所有网页为 N 个，平均每个网页长度为 L 个词。这样词频为 $K/(N \times L)$，文档频率为 $1/N$，注意文档频率和词频在计算上的区别。

布尔检索模型在下一节详细介绍，这里认为一个关键词只记录是否出现在文档中，而不记录其出现的具体位置。即在如图 5-6 所示的倒排索引中，不考虑 NHits 和 HitList 字段的信息。接下来对倒排索引长度的估计中，默认倒排索引不包含 Nits 和 HitList 字段。

在仅符合布尔检索的条件下，对倒排索引的长度进行粗略估计。假定每个文档的平均词汇数为 1 000，则第 i 个最经常使用的词汇在文档中期望出现的次数为 $1\ 000 * 1/13i = 76/i$。

回顾一下文档频率的计算，如果一个词汇在文档中出现 1 次以上，则在计算文档频率时该文档将被计数。文档频率的计算不考虑一个词在文档中出现多少次，而只需要出现即可。例如在 4 个文档中，“搜索引擎”出现在 3 个文档中，则“搜索引擎”的文档频率为 3/4=0.75。第 1 个最经常使用的词汇在文档中出现了 76 次，从期望值的角度看，必定出现在所有的文档中；第 2 个最经常使用的词汇，在文档中平均出现 76/2=38 次，同理也几乎必然出现在所有的文档中。因此得出这样的结论，即在字典中词频排名 1 名~76 名的词汇几乎出现在所有的文档中；词频排名 77~152 的词汇几乎出现在一半的文档中。这不难理解，如果将海量文档两份合为一份计算的话（每个文档的词汇数为 2 000），用前面方法不难得到词频排名 77~152 的词汇也是几乎必然出现，粗略地认为出现在一半的文档中。这样我们得到下面的一个关系，如图 5-8 所示。

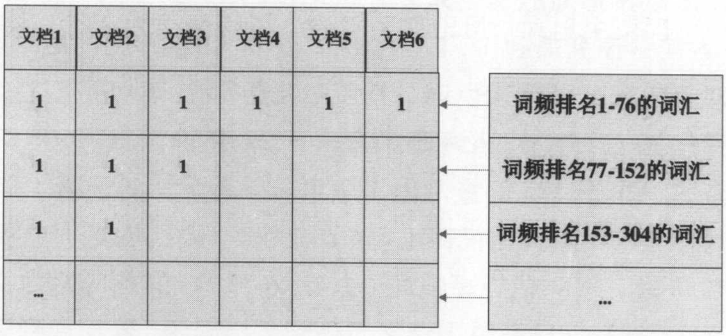


图 5-8　不同词频词汇的文档频率关系

在图 5-8 中用 6 个文档代表全部文档，标“1”表示词汇在文档中出现。可以看出词频排名 1~76 的词汇出现在全部文档中，词频排名 77~152 的词汇出现在 1/2 的文档中，词频排名 153~304 的词汇出现在 1/3 的文档中。

对于共计 500 000 个词汇，则有 500 000/76=6 500 这样的块。对于第 i 块，其出现在 n/i 个文档中。假定记录每个 DocId 需要 k 个字节，第 i 块（含 76 个词汇）需要的空间为 76 kn/i，这样的块共有 6 500 个。不妨假设有 100 亿（10 GB）的网页，每个 DocId 的评价编码长度为 1 个字节（这里采用差分序列对 DocId 压缩编码，实际平均字节数略大于 1）。例如，在倒排索引中对词频排名为 1~76 的高频索引词，其倒排索引简化为如图 5-9 所示。

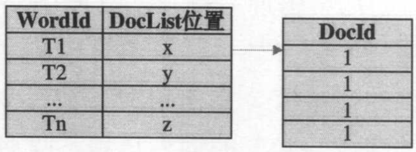


图 5-9　高频索引词在倒排索引中每个 DocId 的差值均为 1

在图 5-9 中由于高频索引词（例如 T1）几乎出现在所有的文档中，所以其指向的文档列表（doclist）几乎为 1，2，3，4，…，n，变为差分序列得到 1，1，1，1，…，1。由于序列间距足够小，因此对于大部分高频索引词，其所对应的每个 DocId 在压缩后所占的空间为 1 个字节。粗略地计算一下，采用 Variable Byte Coding 编码方式，小于 128 的数可用 1 个字节来表示。对于第 1 档高频词（共计 76 个），文档间距（gap）均为 1，因此需要 1 个字节存储文档间距；第 2 档的高频词（共计 76 个），文档间距为 2。由于小于 128，所以只需要 1 个字节存储文档间距。依次直到第 127 档的高频词，均采用 1 个字节来编码。这样共计 127×76=9 652 个高频词，均用 1 个字节表示。剩余的低频词出现在极少的文档中，因此可以认为平均每个文档间距的存储接近 1 个字节。

综上，在布尔检索模型下的倒排索引规模为：

$\sum_{i=1}^{6500} 76 \times 10\text{G} \times 1/i = 760\text{G} \times H_{6500} \approx 760\text{G} \times \ln(6500) \approx 6.7\text{TB}$ 一般情况下，我们可以得出这样的经验公式，即 1 MB 的文档大约需要 1 GB 的索引。为 100 亿（10 GB）网页创建倒排索引（不记录关键词出现的位置信息 HitList，只记录是否出现），大约需要 10 TB（1 TB=1024 GB）的索引空间。如果还需要存储位置信息（HitList），则需要的存储代价更加巨大。一般认为包含存储 DocId 和 HitList 信息后，索引的大小依然是 10 TB 这个数量级。如果按一台 100 GB 硬盘的服务器来计算，至少需要 100 台以上的服务器来做索引。而且为了一些安全稳定的因素，实际需要的服务器近千台。通过以上计算可以得出，索引主要面临的首要问题是“存得下”，下面我们将通过一些技巧来介绍如何存得下如此多的索引。

05 第六节
涉及存储规模的一些计算

前面提到对于索引目前 100 亿中文网页进行倒排索引需要 TB 级的存储空间，这么大规模的数据如何生成？在生成后如何存储？存储后如何支持高效的检索？这一连串的问题将在本节解答。

正排表与倒排表的合并

下载系统将抓取网页存放在网页库中，分析系统在分析后得到网页对象发送给索引系统，因此索引不停地得到这样的网页对象。由于网页对象在分析系统中进行了分词计算，最终分析的结果发往索引系统。

在索引系统中，通过计算不难得到如图 5-10 所示的正排表。在实际操作中，正排表不写入文件，而是保存在内存中。可以理解为内存中存储的正排索引，是正排索引的一种具体表现形式，同理，倒排表也是内存中存储的倒排索引，也是倒排索引的一种具体表现形式。

LId	WordId	NHits	HitList
Docx	T1	m	Hit1,...Hitm
	T4	n	Hit1,...Hitn
	NULL		

图 5-10　正排表

图 5-11 所示是一个倒排表，它们如何合并呢？

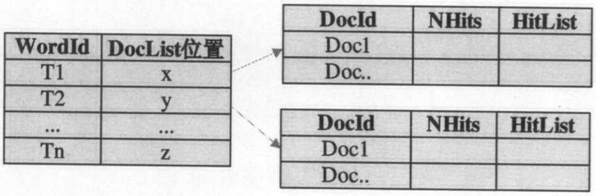


图 5-11　倒排表

正排表与倒排表合并的结果如图 5-12 所示。

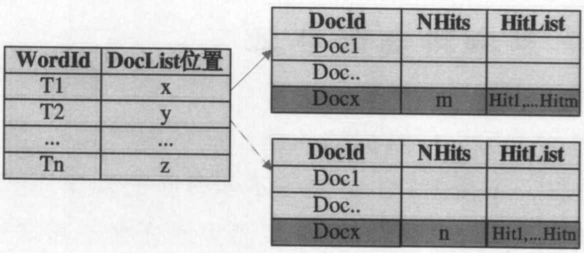


图 5-12　正排表与倒排表合并的结果

如图 5-12 所示，T1 是文档 Docx 的一个词汇。在左边的词典中找到 T1 的位置，然后通过其 Doclist 位置域存放的 DocList 位置信息找到记录表。并将文档编号（DocId）、在文档命中的次数（NHits），以及命中的位置列表（HitList）作为倒排表中的记录表中的一个记录。可以认为正排表和倒排表合并的过程，就是正排表中的数据追加倒排表的数据过程。追加后，正排表并不保留。

读书笔记

而倒排表在内存中存储一定的记录后，成批顺序地写入磁盘，成为临时倒排文件（本章约定，在提到正排表时，表示其存放在内存中；而特指倒排文件时，表示其存放在磁盘中），如图 5-13 所示。

WordId	DocList位置
T1	x
T2	y
...	...
Tn	z

DocId	NHits	HitList
Doc1		
Doc..		
Docx	m	Hit1,...Hitm
Doc1		
Doc..		
Docx	n	Hit1,...Hitn

图 5-13　临时倒排文件

众所周知，由于磁盘的存储特性，所以很难在较大文件的中间追加数据。追加数据就不得不进行大量的数据移动，这种开销是极大的。回到这个例子中来，如图 5-13 所示的临时倒排文件中无法再追加有关 T1 的记录。

综上所述，生成倒排文件之前，倒排表（图 5-12）由于存放在内存中，因此可以任意追加数据。在顺序写入磁盘成为倒排文件后，倒排文件不再变化。但由于不断有新的数据需要进行索引，所以这样的临时倒排文件数量不可避免地还会不断增加。在导入全部需要作索引的数据后，索引系统会有多个这样的临时倒排文件，不妨假设共有 64 个这样的临时倒排文件。

出于性能上的考虑，在一次检索时只需要读取一个倒排文件即可得到全部与检索词相关的索引信息。而不是依次读取全部 64 个临时倒排文件，然后进行结果组合。因此必须将这 64 个临时倒排文件归并成一个更大的倒排文件，称为“最终倒排文件”，其大小略小于 64 个临时倒排文件之和。因为在这 64 个临时倒排文件

中分别保留了词典信息，所以这部分冗余的数据在归并成最终倒排文件后只需要保留一份词典即可。

在具体实现上，还有一种合并方法，简单说来就是将一个正排表（包含多个 DocId 的记录）通过对索引词排序的方法，然后一次性或者分批次地写入倒排表中。在第七节中会提到这个方法，我们这里先通过一个例子理解这个方法。首先正排表的结构需要进行调整，如图 5-14 所示。

LId	WordId	NHits	HitList
1	T1	1	1
1	T2	2	3,6
1	T3	1	7
2	T2	1	1
2	T3	1	3
3	T2	1	1

图 5-14 支持排序的正排表

这里的正排表和前面介绍的稍有不同，可以看到 LId 被冗余地存放。

在内存中保留一块这样的区域存放正排表，每增加一个需要索引的文档，就在正排表中追加一条记录。当追加足够多的记录后，正排表足够大，并符合批量做倒排表的条件后，按照关键词编号（WordId）进行一次稳定排序（稳定排序可以参考有关数据结构的书籍，例如归并排序就是一种稳定排序），得到如图 5-15 所示的排序后的正排表，这里认为 $T1<T2<T3$。

读书笔记

LId	WordId	NHits	HitList
1	T1	1	1
1	T2	2	3,6
2	T2	1	1
3	T2	1	1
1	T3	1	7
2	T3	1	3

图 5-15　排序后的正排表

读书笔记

在图 5-15 中，全部正排表记录按照 WordId 排序。由于采用稳定的排序，所以 LId 的顺序没有被破坏，图中 T2 关键词对应的 LId 依然是 1，2，3。由于在倒排表中 WordId 是顺序存放的，因此排序后的正排表可以依序逐个写入倒排表中，如图 5-16 所示。

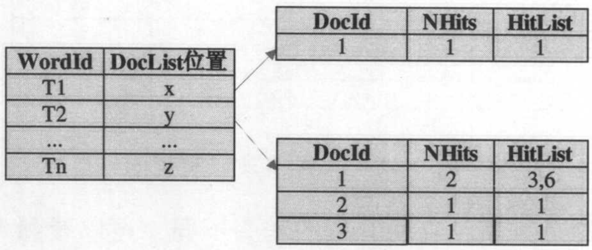


图 5-16　排序后的正排表逐个写入倒排表中

通过估计，每个临时倒排文件的大小大约为 100 MB，因此甚至可以不需要写入倒排表，而直接顺序写入磁盘中的临时倒排文件。这样就形成了内存中存放正排表，磁盘中放临时倒排文件的结果。

多个临时倒排文件的归并

首先考虑两个临时倒排文件的归并，通过前面数据规模估计，

我们知道全部索引大小为 TB 量级的数据，后面将介绍倒排文件如何进行分布式存储。存储的节点数目控制在百这个数量级上，因此全部索引大小分布在 100 个索引结点上，平均每个索引结点大约需要存放 10 GB 的倒排文件（降两个数量级）。如果每个索引结点存放 64 个临时倒排文件，这样每个临时倒排文件约 100 MB（降两个数量级），64 个 100 MB 大小的临时倒排文件最终归并成一个 10 GB 大小的最终倒排文件是十分有挑战性的。每个临时倒排文件内部字典中词汇的编号是有序的，每个词汇对应的记录表（posting list）中的 DocId 是有序的，每个 HitList 中也是有序的，因此归并后依然要保持这种有序性。我们将这种归并过程笼统地称为“归并排序”，很显然这种规模的归并排序在内存中无法完成，下面介绍两种解决这个问题的方法。

（1）拉链法（Zipper）和二路归并。

（2）拉链法和多路归并。

无论是两路归并还是多路归并，拉链法都是必需的。我们首先考察拉链法和二路归并的组合，对于两个较大的文件进行归并排序，不可能将两个文件同时读入到内存中后进行排序。因此可以读取一部分归并，将结果以文件的形式写入磁盘，然后继续读取两个待归并文件的一部分。周而复始，直到读完其一种的某个文件，另一个文件顺序写入结果文件。因此归并的方法如下。

（1）从头开始读取两个临时倒排文件的一小部分（例如每次读取 10 MB）。

（2）分别对 DocId 进行解压，将压缩后的差分序列还原成

原始的差分序列。

（3） 两个按照 DocId 有序的列表进行归并。

（4） 归并的结果进行压缩。

（5） 写入归并后的临时倒排文件 1&2 中。

以上方法如图 5-17 所示。

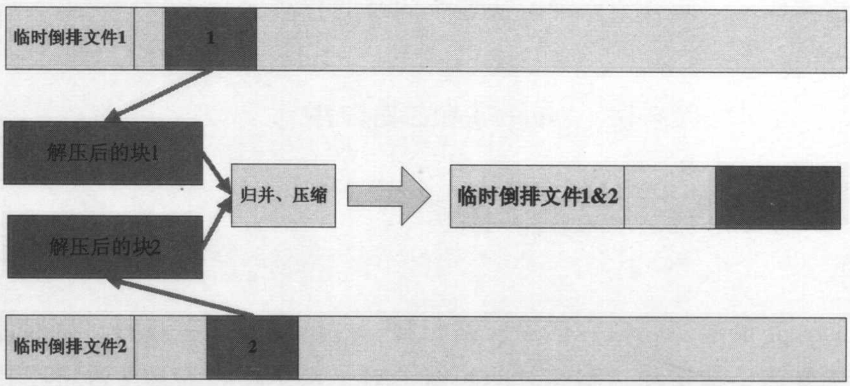


图 5-17 Zipper 方法

这里首先解压临时倒排文件 1 和临时倒排文件 2 的两块数据（只解压 DocId，不解压 HitList 的部分），分别解压后得到解压后的块 1 和解压后的块 2。因为 DocId 有序，因此归并可在线性时间内完成（参考有关数据结构的书籍中关于两个有序表归并的内容，这里不再展开）。归并后的倒排表中 DocId 依然有序，接下来还需要采用 Variable Byte Coding 编码方式对 DocId 进行压缩，最后写入结果倒排文件中（如图 5-17 的倒排文件 1&2）中。

这种拉链的方法采用的思路是处理大文件（Big File）的一种通用思路，即每次仅取出大文件的一部分在内存中进行计算。计算结束后存储计算结果，并释放内存。继续读取文件的下一部分到内存、计算、存储计算结果并释放内存，周而复始直到处理全部数据。

64 个这样的临时倒排文件进行归并需要多少趟这样的两两归并呢？第 1 趟，64 个 100 MB 的临时倒排文件两两归并得到 32 个 200 MB 的临时倒排文件；第 2 趟，32 个 200 MB 的倒排文件两两归并，得到 16 个 400 MB 的临时倒排文件，直到最后剩下两个 3 200 MB 大小的临时倒排文件两两归并得到 1 个 6.4 GB 大小的最终倒排文件，这样整个归并的过程结束。每一趟归并结束后，归并段的个数少一半。

一般情况下，m 个初始归并段采用 k 路归并，则归并趟数为 $\lceil \log_k m \rceil$。如果 64 个初始归并段采用 2 路归并，则需要 6 趟（$\lceil \log_2 64 \rceil$）。由于每趟都不可避免的将全部文件换入内存和换出磁盘的过程，生成一个最终倒排文件的过程中需要产生大量的临时文件。大量的内存和磁盘数据的换入换出，所以整个索引制作的瓶颈主要在这里。通过下面这个例子来直观地认识归并的趟数对索引的巨大影响，如图 5-18 所示。

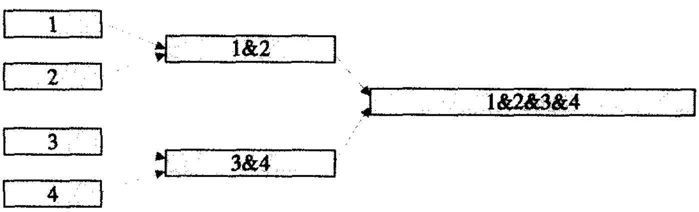


图 5-18 二路归并

图中假定有 4 个临时倒排文件，并且为 100 MB，在第 1 块和第 2 块进行归并时，需要读取的磁盘量为 200 MB，写入量也为 200 MB（块 1&2 的大小为 200 MB）。因此第 1 趟的归并（第 1 块与第 2 块归并，第 3 块与第 4 块归并）共计需要的磁盘 I/O 数量（读写量）为 800 MB。不难计算，第 2 趟归并的磁盘读写量也必然是 800 MB。实际上从宏观上看，每次都要读出全部的内容，计算。然后写入全部的内容，因此每趟的读写量是全部归并段的总数的两倍。这样 64 个 100 MB 大小的临时倒排文件进行 6 趟归并，需要的读写数量为 64×100 MB×2×6，约合 77 GB。

基于前面的分析，对于 m 个初始归并段，外排时采用 k 路归并，则归并趟数为 $\lceil \log_k m \rceil$。显然，随着归并路数 k 的增大，归并的趟数将减少。趟数的降低将大大减少磁盘 I/O 的数量，这样很自然地想到使用多路归并的方法。对于 64 个临时倒排文件的归并，如果能同时归并 64 路（k=64），则只需要一趟归并即可。

多路归并的方法在与数据结构相关书籍中都有介绍，通常采用"败者树"这样的数据结构，当然同时也需要采用拉链的思想。每次读取这 64 个临时倒排文件的一部分进行多路归并，直到处理全部的临时倒排文件，结束归并过程，这里不再展开实现细节。

二路归并在实际处理中还有很多细节，例如每次解压出来的块中并不是全部都能够被归并写入结果倒排文件中会有一些块尾部分；例如解压后的块 1 的尾部包含了编号为 Tx 词汇及其对应的记录表（posting file），而在解压后的块 2 中没有包含编号为 Tx 这个词，那么这部分将需要等待下一次归并才能写入磁盘。问题是一次解压多少大小才使最合适，以及如何进行并发的归并操作等。

针对这些实现的细节，读者可以写一些模拟程序来体验大规模数据归并的全过程。另外可以参考[Managing Gigabytes]书籍官方主页中的一些及有价值的源代码。

倒排索引分布式存储

通过前面的学习，我们知道索引数据的规模为 TB 级。TB 相当于 1 000 GB，一个 1 000 GB 的文件是不可想像的。因此将全部索引文件存放在一台主机上，不仅是不合适的，而且是不安全的。这样一旦这个倒排文件损坏，全部服务就会受到很大影响，因此倒排索引的分布式存储技术应运而生了。

目前倒排索引分布式主要有两种方案，这里假定分布式的方法采用多索引结点（可以理解为多主机）的方法。即把一个巨大的倒排文件通过一些划分方法进行切分，使得每一个索引接点上只保留倒排文件的一部分。这样一方面加快了倒排文件的创建速度，降低了倒排文件损坏带来的损失；另一方面也提高了检索的效果。

多机分布式索引一般按照文档编号（DocId）或者按照索引词编号（WordId）进行划分。按照 DocId 划分的结果称为“局部倒排文件”（Local inverted file）；按照 WordId 划分的结果称为“全局倒排文件”（Global inverted file），如图 5-19 和图 5-20 所示。

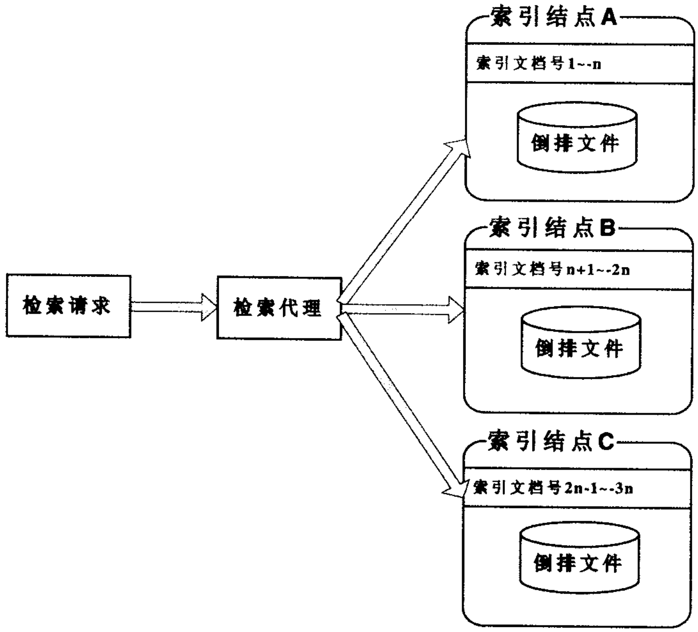


图 5-19 局部倒排文件

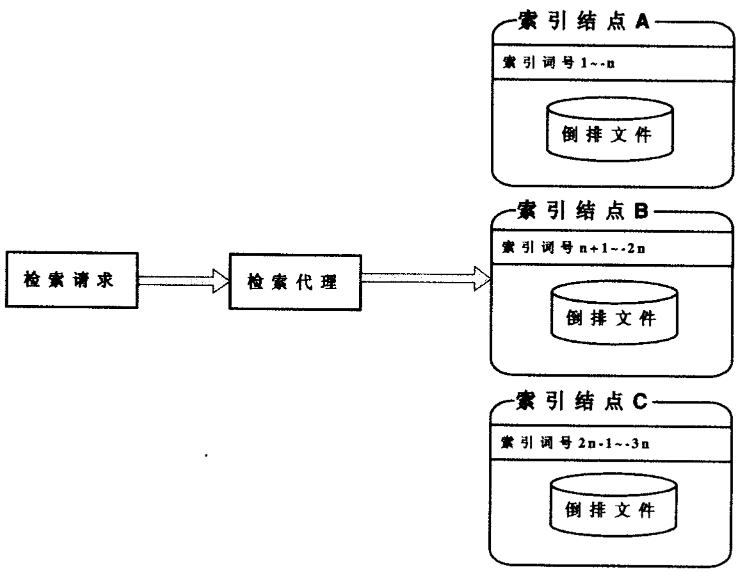


图 5-20 全局倒排文件

图中每个索引结点可以理解为一台独立主机。由于索引被分布式地存储到不同的索引结点上，所以全局还是局部是相对于索引词来说的。对于局部倒排文件与索引词相关的一部分，DocId 被存放在一个索引结点的倒排文件中。换句话说，在图 5-19 中索引结点 A 中的倒排文件只存放了某个关键词的一部分匹配的 DocId；而全局倒排文件则存放了一个关键词全部匹配的 DocId。为了便于表述，以下称这两种方案分别为“局部方案”和“全局方案”。

对于局部方案，索引结点按 DocId 的不同将倒排文件分布式地存储在不同的索引节点上。每个索引结点负责对一个 DocId 区间的文档进行索引，因此每个索引结点的倒排文件中的 DocId 互不相同。在检索时，将检索请求广播到每个结点。每个结点分别查询，最后由检索代理合并查询结果。

对于全局方案，索引结点按索引词 ID 的不同将倒排文件分布式地存储在不同的索引节点上。每个索引结点负责对一个索引词 ID 区间的文档进行索引。因此每个索引结点的倒排文件中的索引词 ID 互不相同，然而每个索引结点的文档可能重复。如果某一个文档的索引词足够多，以至于能够覆盖两个以上的索引词 ID 区间，则可能会被存放在多个索引结点上，这种重复存储将不可避免。

全局方案带来的好处是如果只检索一个单词，那么只需要在一个索引结点中检索即可。在图 5-20 中，对于一个检索请求，发现这个检索词在索引结点 B 中索引。因此整个检索只在索引结点 B 中完成，这就大大节约磁盘 I/O。此外，由于查询可能是不同的查询词，因而被分布在不同的索引结点上。这样并发的用户查询不需要在检索代理中排队，可以并发地查询从而提高效率。例如查询“XML”时，检索代理检测发现这个词存放在索引结点 A 上；

查询“NJU”，检索代理发现这个词存放在索引结点 B 上，因此这两个关键词的查询可以做到并发查询。如果把索引看做某种公众服务，则全局方案是 64 个窗口同时对外服务，而局部方案是单窗口排队服务。

文献[Melnik et al. 2000]阐明了采用局部倒排文件的方案是相对有利的，主要是以下两点。

（1） 可靠性高。

（2） 降低网络负载，提高查询效率。

首先，对于全局方案，如果某个索引结点出现故障，可能导致某一些关键词无法查询；而局部方案在这种情况下（某个索引结点出现故障），最多是损失来自于一个 DocId 区间内的文档。对于查询效果的影响不大，因此该方案提供了足够的可靠性。

其次，对于全局方案，由于所有的查询结果来自于一个索引结点，因此检索代理要等待这个索引结点传输全部的查询结果，这是低效的。而局部方案中多个索引结点同时将查询结果并发传送，从而充分利用了网络带宽。有意思的是局部方案有利于并发地获取检索结果；而全局方案有利于查询并发，这一点请读者细细体会这种平衡的奥妙，因此局部方案在获取查询结果方面是有优势的。

到底哪一种方案最佳呢？在业界，一般认为局部方案的可靠性是必须的，因此主要应用了该方案；而在研究界，有研究表明

[B2S Jeong et al 1995]，在多处理器多磁盘系统下，如果检索词均匀地被请求或者索引词分布偏差不大的情况下，由于避免了局部方案中检索请求必须排队的弊端，因此全局方案在性能上是最佳的。

倒排文件缓存

一般认为一个词被查询的频率与其被使用的频率相当，即频率高的词往往也是查询的热词，查询的频率依然符合齐普夫法则。即查询频率排名为 i 的关键词，其查询的实际频率与 1/i 成比例。大量的实验科学证明，在一段时间内那些有机会被检索到的检索词总是少数的，将这些少数的检索词存放在内存中可以大大降低读取磁盘中倒排文件的机会。关于倒排文件的缓存，可以参考文献[李晓明 2004]。这里只给出一个结论，如果一个索引结点需要 10 GB 的倒排文件，那么在这个 10 GB 的倒排文件中，只有不到 20%的索引词及其记录表应该进入缓存。然而这 20%的索引词占用的空间几乎是 80%，即需要 8 GB 的内存，这显然是难以实现的。因此业界采用了很多特有技术来完成这个工作，由于超出本书的范畴，所以不再深入下去。

倒排文件的缓存及第六章中提到的搜索结果页缓存的基本原理大致相当，读者可以参考第六章中的相关内容，深刻地理解在查询系统和索引系统这种缓存机制的重要性。

倒排索引词典统计信息的计算

倒排文件中的词典还需要有关每个索引词的统计信息，主要

是词汇出现的文档数。这些信息主要用在查询系统中，在下一章中会详细介绍这些统计数据是如何应用的。

在索引系统中，这些关于索引词出现的文档数的统计是在查询请求发生之前预先计算好的，是倒排表的词典部分中不可分割的一部分。把做统计工作的这个模块称为“统计员”。关键词的文档频率的统计信息是全局的，因此在整个系统中仅有一个服务模块来完成这项工作。在系统结构图中，统计员的位置如图 5-21 所示。

如图所示，统计员把各个索引结点的词典信息综合起来做全局统计。然后将统计结果传回各自的词典中，继而保存这些全局统计信息。在下一节说明倒排索引创建过程时会详细提到关于统计信息的计算过程。

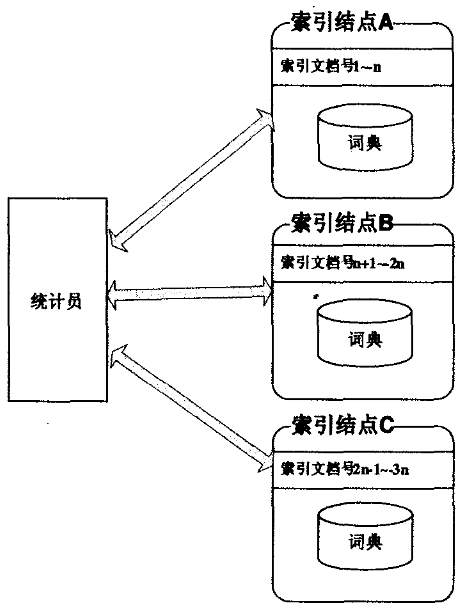


图 5-21　统计员的位置

第七节 倒排索引文件的创建过程 05

倒排索引文件的创建过程更像是一个工程建设，其中大量应用了批量计算及流水计算的技巧，完成如此大规模的倒排索引文件的创建一直是搜索引擎的核心难点。

创建倒排表

首先通过一个例子完整地体会单个索引结点倒排文件的创建过程，如图 5-22 所示。

这里的网页库支持顺序访问模式（参见第三章第五节），能够顺序读取存放在其中的文档。为了简化，我们假定读取的文档按照顺序分别为文档 1、文档 2 和文档 3，其正文内容分别为“rat dog”、“dog cat”和“rat dog”。通过在内存中完成正排得到索引词出现的文档和位置信息，例如，rat（1,1）表示在文档 1 的第 1 个位置出现“rat”这个索引词。接下来通过对字母排序（汉字可以按照汉字词汇编号排序），得到一个临时的按照索引词有序的结构，这有助于顺序写入各个索引词对应的记录表。在图 5-22 所示

的倒排表达到一定大小（例如 100 MB）时，将倒排表顺序写入到临时倒排文件中。完成全部网页库的索引工作后，将产生的多个临时倒排文件归并为一个最终倒排文件（图 5-22 中忽略临时倒排文件归并的过程）。

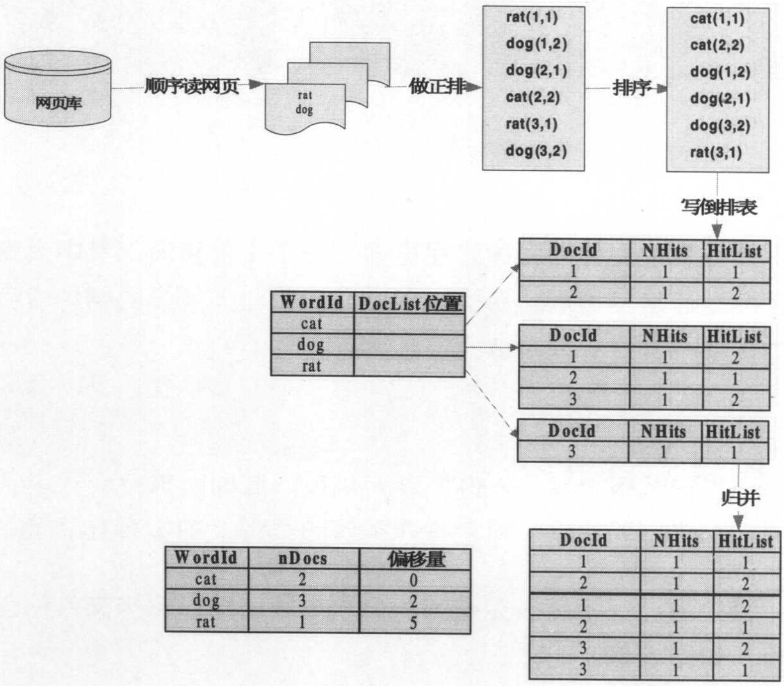


图 5-22　单个索引结点倒排文件的创建过程

从计算的角度上讲，临时倒排文件创建的全过程包含了磁盘读取（loading）、计算（processing）和写入磁盘（flushing）这 3 个过程，磁盘读取从网页库中读取一个个的文档；计算过程包括了正排计算、排序及归并等；写入磁盘主要是写入临时倒排文件。如果采用多道并行处理，则可大大提高索引创建的效率。

如果将整个临时倒排文件创建过程抽象为 L（loading）、P

(processing)和 F(flushing)3 个顺序的操作，即到写入临时倒排文件为止，则 L 操作和 F 操作是磁盘密集型操作；而 P 操作是 CPU 密集型操作。为了简化，采用两个线程并发操作的处理方法，如图 5-23 所示。



图 5-23　两个线程并发操作

这里线程 1 首先开始执行 L 操作，在执行 P 操作时磁盘空闲，这时线程 2 使用磁盘执行 L 操作。理论上最佳的效果是无论在什么时间点上，总是一个线程占用 CPU；另一个线程占用磁盘，这样相互配合可以高效地完成索引创建过程。这里还有一个问题就是一次 L 操作读取多少文档才是最佳的，可以使得这种合作能够最少出现互相等待的情况。对于两道线程并发，一个 L 操作读取的文档规模应该满足。使得 P 操作执行的时间与执行一次 L 操作的时间相当，即线程 1 的 P 操作能够和线程 2 的 L 操作重叠。当然实际处理中还有很多技巧，读者可以参阅文献[Arvind Arasu et al. 2001]获取更加详细的技术细节。

计算统计信息

计算统计信息在上一节中提到，这里给出两种计算方法，两种方法各有优劣。第 1 种方法从排序后的正排表开始统计；第 2 种方法从临时倒排文件统计。分别来看这两种方法的区别。首先通过图 5-24 来理解第 1 种方法。

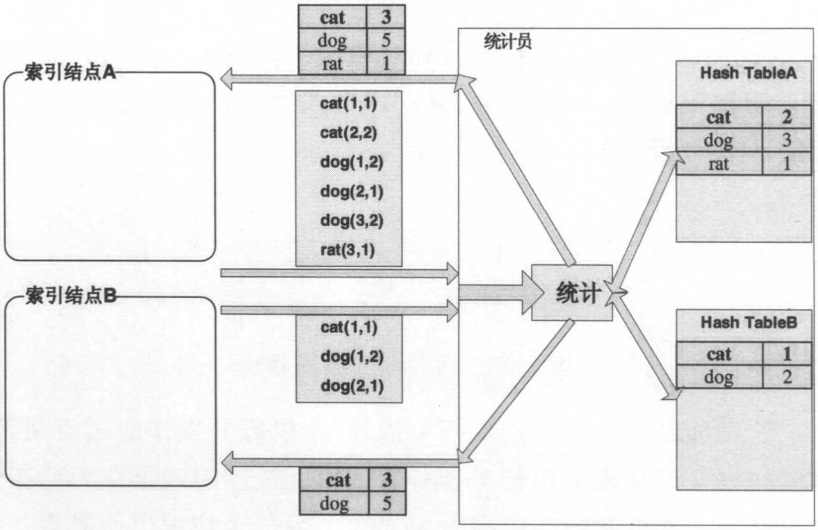


图 5-24　基于全局索引词出现文档数统计方法

内存中经过排序的正排结果在转换为倒排表之前，发给统计员一份拷贝。统计员为每个索引结点建立一个哈希表，这个哈希表用来进行计数。在全部网页库中的文档被处理完后，统计员将各个哈希表中的词进行综合统计，把相应的结果发给各个索引结点。注意这里发给索引结点 A 的统计结果和发给索引结点 B 的统计结果是不同的，因为索引结点 B 不包含“rat”这个索引词，因此没有必要把“rat”的信息发给它。这种方法由于需要维护哈希表的代价，因此需要耗费一定的内存空间，这是其主要缺点。

第 2 种统计方法主要采用基于已经计算好的倒排表数据来进行综合统计，整个过程相对简单。相当于对各个索引结点自身的统计结果进行综合统计，然后回传给各个索引结点。这种方法的主要缺点是需要等待最慢的索引结点做完索引后才能开始进行计算。

在完成了创建最终倒排文件和词典后，全部倒排索引文件创建工作完毕。从某种角度上看，这些都是一种预先计算（precomputation）。这种预先计算都是在为查询时节省时间，海量数据完成一次最终倒排索引文件的制作是非常耗时的，这些尽可能预先完成的计算为查询争取了宝贵的时间。

现在离搜索越来越近了，下一章我们来到搜索引擎最直接面对用户的查询系统，继续了解有关搜索及查询的知识。

参考文献

[1] [Arvind Arasu et al. 2001] Arasu, A. and Cho, J. and Garcia-Molina, H. and Paepcke, A. and Raghavan, S. Searching the Web, ACM Transactions on Internet Technology , 42 pages

[2] [B2S Jeong et al 1995] B2S Jeong ,E Omiecinski . Inverted file partitioning schemes in multiple disk systems . IEEE Transactions on Parallel and Distributed Systems

[3] [S. Brin 1998] S. Brin and L. Page. The anatomy of a large-scale hyper textual Web search engine. Computer Networks and ISDN Systems, 30 (1-7) :107-117 , 1998

[4] [Melnik et al. 2000] MELNIK, S., RAGHAVAN, S., YANG, B., AND GARCIA-MOLINA, H. 2001. Building a distributed

full-text index for the web. In Proceedings of the Tenth International Conference on The World-Wide Web.

[5] [Witten 1994] I. H. Witten, A. Moffat, and T. C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images. New York, NY: Van Nostrand Reinhold , 1994

[6] [Managing Gigabytes] http://www.cs.mu.oz. au/mg/

[7] [Williams 1999] H. E. Williams, J. Zobel. Compressing Integers for Fast File Access. The Computer Journal, 1999, 42 (3)

[8] [Zipf 1949]Zipf G K. Human Behavior and the Principle of Least Effort . New York :Addion2Wesley , 1949

[9] [Stanford IR]http://www.stanford.edu/class/ cs276a/

[10] [李晓明 2004] 李晓明，闫宏飞，王继民，"搜索引擎——原理、技术与系统"，科学出版社

[11] [姜望琪 2005] 姜望琪 "Zipf 与省力原则[J]"，同济大学学报:社科版，2005,16 (1) :87-94

第六章

搜索引擎的查询系统

06 第一节 知识准备

在搜索引擎 4 大系统中，第 4 个系统称为“查询系统”。查询系统直接面对用户，在接收用户的查询请求后，通过检索、排序及摘要提取等计算，将计算结果组织成搜索结果页返回给用户。整个查询过程不仅要快，而且必须能够提供用户满意的查询结果。文献[S. Brin 1998]提到“The goal of searching is to provide quality search results efficiently.”因此，本章从效率和效果这两个角度进行探讨，首先一起来了解一些关于查询系统的常用术语。

什么是信息熵

信息是个很抽象的概念，直到 1948 年，香农提出了“信息熵”（shāng）的概念才解决了信息的量化问题。

回忆一下数据结构中介绍过的哈夫曼编码，该编码通过计算不同词汇的词频并依据大小关系构建哈夫曼树，通过哈夫曼树为不同词频的词汇创建不同长度的前缀编码。从直观上看，高词频词汇编码较短，低词频词汇编码较长，这些编码都是 0 和 1 组成

的比特串。直观的感觉还难以揭示事实的真相，下面通过一个例子来揭开编码长度和概率之间的联系。

假定红军和蓝军进行战术演习，红军打算左右夹攻蓝军，那么为了使得红军的兄弟部队能够相互通信，需要事先商量进攻的口令。并且由此确定是否同时发起攻击，取得更大的战果。假定有如下 3 类通信口令。

（1） 如果太阳围绕地球转，就发动攻击。

（2） 要么在白天攻击，要么在黑夜攻击。

（3） 在 0 点~6 点或者 6 点~12 点或者 12 点~18 点或者 18 点~24 点攻击。

不考虑信息安全的情况下，红军的两支部队需要如何准备这些消息编码呢？显然编码越短，越有利于战场恶劣的条件，并节约通信成本。下面就对这 3 种情况分别进行分析。

对于第 1 种情况，红军两支部队不需要任何通信。因为太阳围绕地球转，这是肯定的。通信代价为 0，这样无论两支部队选择何种攻击方式都是合理的。

对于第 2 种情况，红军两支部队需要约定通信方式。假定为红 A 部通过通信网络传送一个比特 0，表示在白天发动攻击；传送一个比特 1，表示在黑夜发动攻击。显然，此时的通信代价为 1 个比特，1 个比特能够表达两种可能性。由于红军两支部队保证

在白天或者黑夜攻击，而不会出现一支部队选择白天，一支部队选择黑夜进行攻击的可能性。因此这种信息更加有价值，然而也付出了 1 个比特的通信开销，不妨认为这个信息“值”一个比特。

对于第 3 种情况，虽然相对复杂，但也可以以如下 4 种编码方式约定口令。

（1） 00：在 0 点~6 点攻击。

（2） 01：在 6 点~12 点攻击。

（3） 10：在 12 点~18 点攻击。

（4） 11：在 18 点~24 点攻击。

此时的通信成本为两个比特，两个比特能够表达 4 种可能。这样红军两支部队的进攻同步性更强，其先后进攻的时间最多差 6 小时，很显然这种信息比第 2 种更加有价值。同样道理，认为这个信息“值”两个比特。

综合上述 3 种情况的分析，从直观上看，信息包含的情况越多，信息越有价值，需要的通信代价就越大。信息是否有价值隐约地和概率有着密不可分的关系，那么如何衡量发起攻击这个信息，如何从直观跨越到客观呢？

1948 年，香农长达数十页的论文“通信的数学理论”成为信息论正式诞生的里程碑。在其通信数学模型中清楚地提出信息的

度量问题，得到了如下著名的计算信息熵（Entropy）公式：

$$H(X)=\sum_{i=1}^{n}-p_i\log(p_i)$$

对于上例中的第 3 种情况，假定每一种攻击可能均等，均为四分之一，那么何时发动攻击这样一个随机变量 X 包含 4 个随机事件，概率分别表示为：

P（X=“在 0~6 点攻击”）$=\frac{1}{4}$

P（X=“在 6~12 点攻击”）$=\frac{1}{4}$

P（X=“在 12~18 点攻击”）$=\frac{1}{4}$

P（X=“在 18~24 点攻击”）$=\frac{1}{4}$

那么

$$H(X)=-\frac{1}{4}\log_2(\frac{1}{4})+-\frac{1}{4}\log_2(\frac{1}{4})+-\frac{1}{4}\log_2(\frac{1}{4})+-\frac{1}{4}\log_2(\frac{1}{4})=2$$

与前面计算的结果 2 个比特一致。

信息熵 H（X）在信息论中称为消息 X 的“熵”，其含义是信

息集 X 发出任意一个随机事件的平均信息量。"熵"值 $H(X)$ 说明了消息集 X 的每个事件的平均存储的位数，即用多少个二进制表示一个消息。在约定进攻口令的这个例子中，4 个口令中发生一个口令平均需要的通信或者存储代价（平均信息量）为两个比特。

香农通过"熵"阐明了概率与信息的关系，即变量的不确定性越大，熵也就越大，将其搞清楚所需要的信息量也就越大。信息熵是一个十分重要的概念，下面在介绍经典的 TF/IDF 方法时还将就此问题继续展开。

检索和查询的区别

本章约定对于查询来说，适用于真实用户进行的一次查询是相对于搜索引擎查询系统而言的；对于检索来说，适用于检索代理对索引库进行的一次检索是相对于搜索引擎索引系统而言的。查询的结果是搜索结果网页，检索的结果是与查询词相关的文档列表（doclist）。

检索词和查询词的区别

严格意义上，普通用户提交给查询系的关键词称为"查询词"；经过查询系统分词，提交检索代理的称为"检索词"。例如用户提交查询词为"清华大学图书馆"，通过分词，提交给检索代理变成"清华大学"和"图书馆"两个检索词。为了简化，本章并不区分查询词和检索词，而统一使用查询词这个术语。

自动文本摘要(Automatic Text Summarization)

自动文本摘要简称“自动摘要”，它是从文档中自动提取出的一个正文片断。用户仅仅需要浏览整个正文片段就能够了解文档中与查询词相关的部分，进而判断是否值得详细阅读整篇文档。

06 第二节 网页信息检索

网页信息检索的数据源来自于网页索引库（在前一章中介绍了网页对象被索引入库的全过程），网页信息检索输出是一组文档编号，这些被编号的文档都是索引库中包含查询词的文档。

早期的检索模型

早期的检索模型是一种称为“布尔模型”（Boolean Models）的检索模型。布尔模型也称为“集合模型”，是一种采用 AND、OR 及 NOT 等逻辑运算符将多个查询词连成一个逻辑表达式，继而通过布尔运算进行检索的简单匹配模型。例如查询词为“走进搜索引擎 检索模型 - 搜索”，将会被翻译成“走进搜索引擎 AND 检索模型 NOT 搜索”这样的逻辑语言。按照自然语言的翻译，这个逻辑语言表示包含“走进搜索引擎”且包含“检索模型”，却不包含“搜索”的文档集合。对于查询系统来说，这样的查询词表示用户请求检索包含“走进搜索引擎”且包含“检索模型”，却不包含“搜索”的文档集合。

布尔模型的这种检索易于实现，检索速度快。但是由于没有考虑文档和查询词的相关性问题，没有区分查询词的权重问题。因此在“效率”和“效果”的两难选择上放弃了“效果”，而仅仅考虑了“效率”。

此外，如果查询词中有一个关键词没有包含，则可能出现漏检。不妨通过一个例子来说明布尔模型的这些主要的缺点。

假定有如下这样 3 篇待检索的文档。

（1） 在传统搜索引擎架构中，搜索引擎由 4 个系统构成，分别是下载系统、分析系统、索引系统及查询系统。

（2） 机械行业内一般把小型挖掘机简称为“小挖”，小挖由 5 个系统构成，分别是……，详细地理解这些名词可以使用 Google 搜索引擎搜索一下。

（3） 搜索引擎有 4 个主要功能模块，分别是下载系统，分析系统，索引系统和查询系统。这 4 个系统是搜索引擎的核心，其中查询系统是搜索引擎惟一直接面对客户的系统。

如果采用布尔检索模型，在查询“搜索引擎 系统构成”这样的查询词时，文档 1 和文档 2 均会被检索到，因为文档 1 和文档 2 均包含了全部查询词。显而易见，在第 1 个文档中，“搜索引擎”这个关键词出现了两次，“系统构成”出现了 1 次；在文档 2 中“搜索引擎”出现了 1 次，“系统构成”也出现了 1 次，直觉上看应该是文档 1 的相关性更好。在布尔模型中很难进行相关性强弱的度

量，它只解决“有”还是“没有”的问题，不解决“好”还是“不好”的问题。

最后，从用户查询意图上看，文档 3 比文档 2 更加符合用户的查询意图。文档 3 中出现了 3 次“搜索引擎”这个关键词，仅仅因为没有包含“系统构成”这个关键词，而没有被检索出，而文档 2 只是沾边提到了搜索引擎，却能够被检索出。

布尔模型归纳起来存在如下两大优点。

（1） 表达简单且易于实现。在关键词检索的过程中，把检索计算转变为集合运算，特别是集合间的求交集运算和集合间的差运算。

（2） 检索速度快。布尔模型的计算主要是集合求交运算，这将在下一节中介绍。

正是由于布尔模型的两个优点造成了布尔模型的如下两大不足。

（1） 如果有一个查询词没有被包含，则检索失败。由于布尔模型表达简单，缺乏灵活性，造成上例文档 3 中没有包含“系统构成”这一关键词，因而无法被检索出来的情况。

（2） 检索出来的结果很难进行相关性排序。由于布尔模型计算简单，例如前面的例子中检索“搜索引擎 系统构成”的过程中，文档 1 和文档 2 与查询词的相关性没有被计算，从而无法了

解哪个文档更加符合用户的查询意图（通常认为符合用户查询意图的文档在搜索结果中应排名靠前）。

布尔模型的不足主要由于没有考虑到关键词在查询中的权重问题，这一点不足在向量空间模型（Vector Space Models）中得到部分解决。虽然不够完美，但是已经足够解决上面例子中的检索问题。

向量空间模型（Vector Space Models）

和布尔模型不同，向量空间模型主要关心的是“效果”，而非“效率”。向量空间模型提出了将查询词和文档按照关键词的维度分别向量化，然后通过计算这两个向量间夹角余弦的方法得到文档与查询词的相似度。从而优先检索那些和查询词相似度大的文档，并且能够对检索出的文档按照与查询词的相似度进行排序。向量空间检索模型的计算方法如图 6-1 所示。

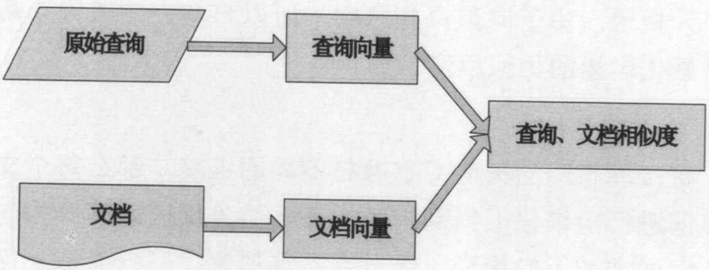


图 6-1　向量空间检索模型的计算方法

在向量空间检索模型中，通过下面 3 个步骤进行检索。

（1） 把原始查询和文档都看做是文本，使用同样的向量化过程分别得到查询向量和文档向量。

（2） 通过计算向量相似度的方法计算原始查询和文档的相似度。

（3） 按照与查询词的相似度从大到小排序文档，返回给用户。

向量（vector）是一个很抽象的概念，它又称为“矢量”。最初被应用于物理学，很多物理量，如力、速度、位移、电场强度，以及磁感应强度等都是向量。大约在公元前 350 年，古希腊著名学者亚里士多德就已提出力可以表示成向量，两个力的组合作用可用著名的平行四边形法则来得到。英国大科学家牛顿最先使用有向线段表示向量。

事实上，向量包含了两层含义，即长度和方向。长度用向量的模表示，向量的模（长度）的计算公式为向量的每个分量的平方和开根号。由于向量具有方向，所以方向上的差异（角度）被用来量化向量的相似程度。

将各种不同的关键词看做是不同的维度，那么每个文档按照关键词进行向量化，得到向量中每一个分量可以理解为向量在各个关键词维度上的投影。这一点不难理解，三维坐标上描述一个点采用的方式为（a，b，c）表示向量在 X 轴上的投影为 a，在 Y 轴上的投影为 b，在 Z 轴上的投影为 c。在这里只是把代表三维空间中 3 个轴转换为 n 个关键词的 n 维空间，这样每一个查询句子和每一个文档都可以用这个 n 维空间来表示。

通过下面的一个例子来理解向量化的过程。假定汉语的词汇表只有“走进”、“搜索引擎”和“学习”这 3 个词（实际上，常用的汉语词汇过万），那么这 3 个词组成的向量空间就是我们熟悉的三维空间，如图 6-2 所示。

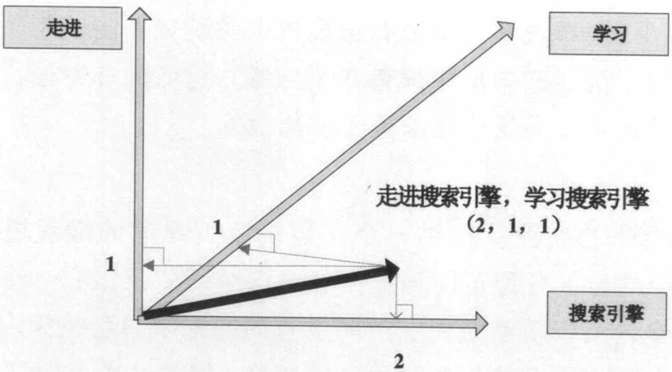


图 6-2　三维空间

在图中，对于“走进搜索引擎，学习搜索引擎”这个句子，通过计算每个词汇的出现的次数，得到这样的统计信息。即“搜索引擎”出现两次，“走进”出现 1 次，“学习”出现 1 次。将 3 个词的维度理解为三维空间的 XYZ 轴，这样“走进搜索引擎，学习搜索引擎”在词汇表构成的向量空间内表示为向量（2，1，1）。这个向量的 3 个分量的意义可以理解为对 3 个轴的投影分别是 2，1，1，物理含义为这些关键词在查询句子中分别出现的次数，同时注意这里向量的方向性用箭头表示。

现在我们扩展到四维空间上理解，假定词汇表中还包括了“检索模型”一词，这样对于“走进搜索引擎，学习搜索引擎”这个句子进行向量化的结果可能是（2，1，1，0），其中四维空间的第四维表示“检索模型”。由于这个句子中没有出现“检索模型”，

因此它在这个关键词维度上的投影为 0。

据统计常用汉语词汇大约 5 000 条，如果用这 5 000 维的词向量空间表示这个句子，可能是这样的形式，即（0，0，….0，2，0，…1，…1，…）。其中标“0”的分量表示句子在这个词汇上的投影是 0，或者说句子中没有出现这个关键词。由于句子中只出现 3 个词，因此在向量中只有 3 个分量为有效的非 0 值。可见在实际的计算中，向量通常都是十分稀疏的。

向量化的过程就是对一个文档按照关键词的维度进行向量化，每个向量的分量可以理解为包含这个词的权重（weight）。出现次数多的词权重就较大，对向量方向的影响力也较大。为了使不同文档和查询词的相关性具有比较性（相关性排序的需要），保证对大文档和小文档做到公平，还需要对关键词的出现次数做归一化的工作，即转化为词频（词数/总词数）作为向量的分量。因此在上面例子中，“走进搜索引擎，学习搜索引擎”中的“走进”的词频为 1/4，“搜索引擎”的词频为 2/4，“学习”的词频为 1/4。因此在如上图所示的关键词向量空间下，这个查询词被向量化为（2/4，1/4，1/4）。

事实上，向量中的每个分量同除以相同的数不会改变向量的方向，但是会改变向量的距离。因此在只考虑向量方向，而不考虑向量长度的情况下，没有必要使用词频作为向量的分量，这样反而引入了浮点计算的麻烦。考虑到其他可能需要进行向量距离运算的场合，以及为下一小节中的 TF/IDF 的权重量化计算做准备，提前了解词频的有关概念，并使用关键词词频作为向量的分量表示。

在向量化的工作完成后（下一节将提到实际上采用经典的TF/IDF 方法进行向量化的工作），就需要解决计算文档和查询词相似度的问题。向量空间模型中一般采用向量之间的夹角余弦值作为向量是否相似的度量依据。

向量间的夹角余弦的计算公式：

$$\cos\theta = \frac{a \bullet b}{|a| \times |b|}$$

其中 a,b 表示向量，$\bullet$表示向量的点乘，$|a|$表示向量的模，或者说是向量的长度。

通过一个具体的例子理解这个计算过程。

假定在一个 7 个关键词的向量空间下，一个查询词向量化为 a（0，0，2，0，1，0，1），一个文档向量化为 b（0，1，3，5，2，4，0），夹角余弦计算方法如下：

$a \bullet b$ =（0，0，2，0，1，0，1）*（0，1，3，5，2，4，0）^{T}

= 0*0 + 0*1 + 2*3 + 0*5 +1*2 + 0*4 + 1*0

= 8

$$|a| = \sqrt{0^2+0^2+2^2+0^2+1^2+0^2+1^2} = \sqrt{6} = 2.45$$

$$|b|=\sqrt{0^2+1^2+3^2+5^2+2^2+4^2+0^2}=\sqrt{55}=7.42$$

$$\cos\theta=\frac{a\bullet b}{|a|\times|b|}=\frac{8}{2.45*7.42}=0.44$$

这样查询词 a 和文档 b 的相关性就转化为 0.44 这个具体的数值上，使得相似性成为可以量化的概念，因此相似性量化的结果称为“相似度”。

在实际计算中，如果向量 $\boldsymbol{a}$ 表示查询向量，向量 $\boldsymbol{b}$ 表示文档向量，在计算查询向量和一组文档向量的相似度时，查询向量总是不变的，或者说对每个文档向量来说查询向量都是相同的。因此相似度计算中是否除以 $|a|$，对将来进行的相似度排序没有影响，可以作为公共因子消去。计算相似度实际只需要计算 $|a|\times\cos\theta$ 即可，方法如下：

$$|a|\times\cos\theta=\frac{a\bullet b}{|b|}$$

其中每个文档向量的模可以预先计算并保存，而不需要每次查询都执行一次文档向量的模运算。这样，每次求相似度只需要一次向量点乘和除法计算即可。

对于两个高维稀疏向量（由于汉语词汇众多，实际向量化后的向量维数高，非 0 值少），向量的表示和向量点乘的计算也是需要一定技巧。可以采用哈希表的方法快速找到两个向量相同分量的非 0 值进行计算，这里不再详细展开。

为了简化描述，在此前提到的关键词量化过程中采用词频作为向量化中每个向量的分量，而事实上却采用了经典的 TF/IDF 方法为每个关键词进行更加合理的量化。下面我们将走进经典的 TF/IDF 方法，领略信息检索的精髓。

关键词权重的量化方法 TF/IDF★

在实际的查询词及文档向量化应用中仅仅使用词频作为分量是不够的，例如我们曾经使用的例子中，查询“搜索引擎 系统结构”无法检索出文档 3。这是因为没有突出“搜索引擎”的重要性，所以还是无法取得良好的检索效果，接下来将介绍搜索引擎中经典的 TF/IDF 权重计算方法。

首先，我们来继续本章开头提到的信息熵这个概念深入理解概率与信息的关系。下面我们科学地给出“自信息”和“熵”的概念。

定义 1（自信息）：任意随机事件的自信息量定义为该事件发生概率的对数的负值，设该事件 x 的概率为 $p(x)$，那么其自信息定义为：

$$I(x) = -\log(p(x)) = \log(\frac{1}{p(x)})$$

自信息也可以理解为某个概率的事件进行编码需要的最小编码长度。

定义 2（熵）：在信息论中自信息量是一个随机变量，它不能用来作为整个信源的信息测度。因此我们引入平均自信息量，即熵，定义为：

$$H(X)=\sum_{i=1}^{n}-p_i\log(p_i)$$

Claude Shannon（香农）的源代码[Shannon,1948]理论指出，最理想的编码方法是词汇表中第 i 个词汇预期出现的概率为 p_i，那么该词汇需要分配 $-\log_2(p_i)$ 个比特长度的编码。最佳编码符号中的比特数目表示符号的信息内容（information content），整个词汇表中的全部词汇的信息量的平均大小称为"概率分布的熵"，即：

$$E=\sum_{i=1}^{n}-p_i(\log_2 p_i)$$

E 用比特/符号为单位表示，表示词汇表平均每个词汇需要的加权平均编码长度 [Baeza-Yates et al,1999]。

在编码中用"熵"值衡量是否最佳编码。若以 W 表示采用一种编码方式后词汇表平均编码长度，则可能情况如下。

（1） $W>H(X)$：有冗余，不是最佳编码。

（2） $W<H(X)$：不可能。

（3） $W==H(X)$：最佳编码（一般 W 稍大于 $H(X)$）。

这里$\sum_{i=1}^{n} p_i \times l_i$　p_i为词汇表中的第 i 个词汇的发生概率(文档频率),l_i为给字母表中的第 i 个词汇的实际编码位数,因此"熵"值是平均编码长度的下限值。数据结构中提到的哈夫曼编码就是对"熵"值的极限逼近,由于实际编码长度不能为小数,因此哈夫曼编码所能达到的压缩比距离压缩极限还有一定距离。

熵最大限度地压缩冗余的信息对于衡量关键词权重具有特殊意义,我们通过一个例子循序渐进地来理解为什么要把熵这个概念引入到关键词权重的量化计算中来。

为了简化,假定汉语的词汇表中总共仅有 3 个词汇,分别为"走进"、"搜索引擎"及"学习",对这 3 个词汇在实际应用中使用的情况进行概率统计,并且分别计算其各自出现的概率为 1/256、1/4 096 及 1/256。因此按照熵的计算公式,汉语词汇表中的词汇平均编码长度为:

$$\frac{1}{256}\log_2(256)+\frac{1}{4096}\log_2(4096)+\frac{1}{256}\log_2(256)=0.0654。$$

接下来,假定一个文档就是一个信息源。这个文档包含了 T_1,T_2,T_3,…,T_n 共 n 个词汇。每个词汇各出现了 N_1,N_2,N_3,…,N_n 次,其中在海量文档中出现的文档频率(词汇的发生概率)分别为 D_1,D_2,D_3,…,D_n,那么传输一个这样的文档需要多少编码长度呢?

假定每个词汇的出现相互独立,并且不考虑出现的先后顺序,因此由这些词汇组成这篇文档的概率为:$X=D_1^{N_1}\times D_2^{N_2}\times\ldots\times D_n^{Nn}$(概率的乘法公式)。参考自信息的公式,对具有这种概率的事件

进行编码，需要的编码长度为：

$$-\log(X) = -\log(D_1^{N_1} \times D_2^{N_2} \times \ldots D_n^{N_n})$$

$$= -N_1 \times \log D_1 - N_2 \times \log D_2 - \ldots - N_n \times \log D_n$$

因此这篇文档理论上能够最大极限地被压缩到$-\log(X)$个比特。数学上可以证明，这样的压缩形式是极限压缩。所有的压缩算法都参考对熵压缩的逼近程度来量化压缩能力的大小，在数据结构中学过的哈夫曼编码就是对信息熵的一种逼近编码。另外上面的公式也可以这样理解，对于关键词 T_i，其文档频率为 D_i。这个关键词的编码长度为$-\log D_i$，在文档中该词出现了 N_i 次，因此总共需要的编码长度为 $-N_i \times \log D_i$。而全部文档需要的编码长度为 $\sum_{i=1}^{n} - N_i \times \log D_i$ ，这与前面的计算结果相同。接下来我们考察该文档中每个词的平均编码长度。

很明显，平均编码长度为全部文档需要的编码长度除以总词数，即：

$$\frac{\log(X)}{\sum_i^n N_i}$$

展开这个计算式得到：

$$\frac{-N_1 \times \log D_1 - N_2 \times \log D_2 - \ldots - N_n \times \log D_n}{N_1 + N_2 + \ldots + N_n}$$

在这个平均编码长度中，各个关键词都做出了不同的贡献。我们将关键词在文章中的重要性量化为对平均编码长度的贡献上，不难得出这样的结论，即越是出现次数多（词频高）且罕见的词汇（文档频率低）对平均编码长度大小的贡献越大。

假定一个文档总词数为 K（$K=\sum_i^n N_i$），对于关键词 T_i 来说，它对平均编码长度做出的贡献为：

$$\frac{-N_i \times \log D_i}{K}$$

进一步转化这个表达式得到：

$$\frac{N_i}{K} \times \log \frac{1}{D_i}$$

其中：$\frac{N_i}{K}$ 为在文档中关键词 T_i 的词频（TF，term frequency），$\log \frac{1}{D_i}$ 为文档中关键词 T_i 的文档频率倒数的对数式，称为“文档频率”（IDF，inverse document frequency）。上面这个平均编码长度的公式可以改写为：

$$\frac{-N_1 \times \log D_1}{K} + \frac{-N_2 \times \log D_2}{K} + \ldots + \frac{-N_n \times \log D_n}{K}$$

显然不同关键词的 TF×IDF 值对编码长度的最终形成做出了不同的贡献，越是出现次数多（词频高）且罕见的词汇（文档频率低）对平均最终计算得到的平均编码长度的大小贡献大，这就

是经典的 TF/IDF 方法。

文档频率是从哪里来的呢？回顾前面搜索引擎索引层的倒排索引词典统计信息中提到的文档频率的计算方法，在索引阶段这些数据已经计算完毕，因此这里可以直接使用文档频率来进行权重计算。对于“走进搜索引擎，学习搜索引擎”这个句子，假定“走进”的文档频率是 1/256，“搜索引擎”的文档频率为 1/4 096，“学习”的文档频率为 1/256。那么它们各自的权重分别如下。

（1） 搜索引擎：$\frac{2}{4}\times\log(4\,096)=6$。

（2） 走进：$\frac{1}{4}\times\log(256)=2$。

（3） 学习：$\frac{1}{4}\times\log(256)=2$。

这样一个句子在仅由这 3 个词构成的向量空间中被量化为（6，2，2），和前面仅使用词频作为量化标准的（2，1，1）相比，突出了“搜索引擎”这个关键词在句子中的重要性。使得特别是一些诸如“我们”及“他们”这样的高频词能够很好地降低权重，而提高了低频词的权重。并且使得具有不同权重的词汇在进行向量相似度计算过程中发挥不同的作用，使得量化的结果更加科学。

搜索引擎采用的检索模型

搜索引擎采用了布尔模型和向量空间模型结合的方法来进行信息检索，布尔模型的检索效率高且易于实现；向量空间模型能够提高检索的相似度，通过相似度排序的手段能够大大改善查询效果。因此搜索引擎将二者的优势相结合，完整的检索过程如图6-3所示。

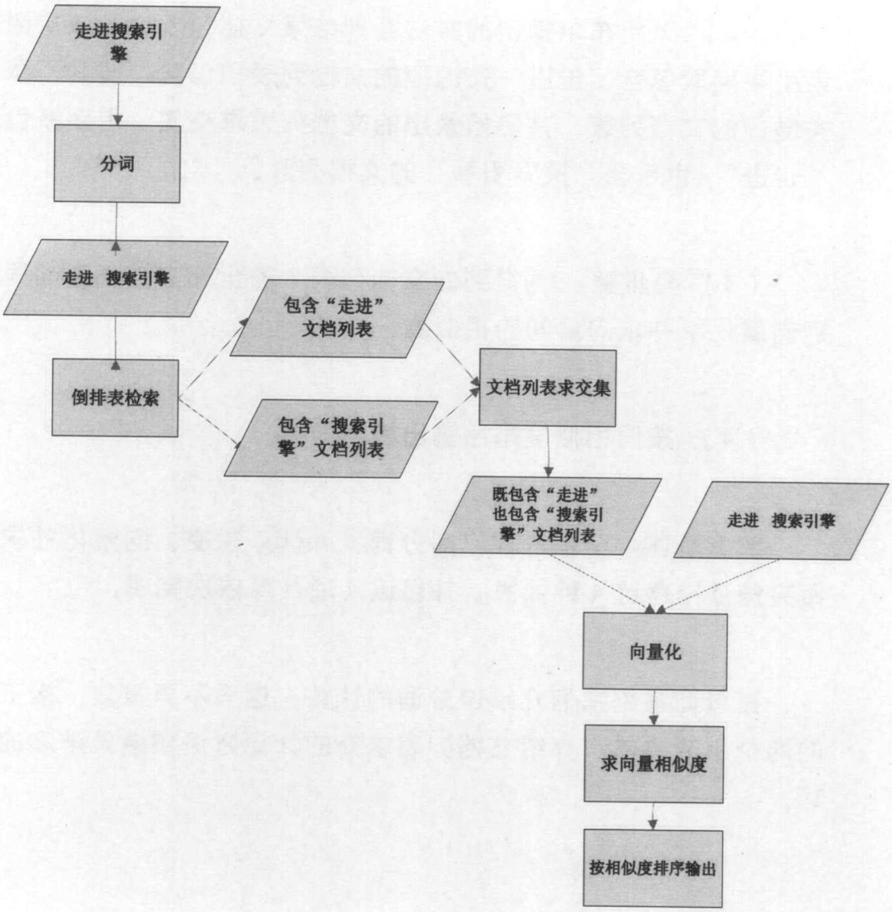


图 6-3　完整的检索过程

读书笔记

如图 6-3 所示，图中方块为计算部分，斜方块为数据部分，详细的检索过程如下。

（1） 对查询词进行分词，得到一个逻辑表达式。例如查询“走进搜索引擎”，将会被切分成“走进”，“搜索引擎”这两个词。并且转换为用 AND 逻辑表示的表达式，即“走进” AND “搜索引擎”。

（2） 采用布尔模型的方法得到结果文档列表，例如从倒排索引中提取包含“走进”关键词的文档列表和包含“搜索引擎”关键词的文档列表。并将检索出的文档列表求交集，得到既包含“走进”，也包含“搜索引擎”的文档列表。

（3） 将步骤（2）得到的文档列表中的全部文档和查询词分别向量化，并求向量间的相似度。

（4） 按照相似度排序输出检索结果。

综上所述，全部过程包括分词、doclist 求交、向量化并求向量夹角及排序这 4 种计算，并且这 4 项计算依次完成。

在查询系统我们介绍过分词的计算，这里不再重复，接下来的两个小节将依次介绍文档列表求交的计算方法和结果排序的技巧。

多文档列表求交计算

在实际的查询中，包含一个或者多个查询词。有时一个查询词也会因为分词而分解出多个词，因此可能包含如下 3 种情况。

（1） 查询单个词：例如查询“中国”。

（2） 查询多个词：例如查询“中国 搜索引擎”，搜索引擎默认查询词中间空格表示用户主动的分词，认为是一次多词查询。

（3） 查询一个词：由于被分词，而成为实际的多词查询。例如查询“走进搜索引擎”将会被分解成为“走进 搜索引擎”，本质上和第 2 种类型相当。

对于第 1 种情况，只需要在倒排索引表中检索出一个关键词对应的文档列表。由于检索结果是单个文档列表，因此不需要进行多文档列表求交的计算过程。

对于第 2 种和第 3 种情况，都需要进行多文档列表的求交计算。例如在倒排索引表中检索出包含“走进”一词的文档列表为 $doclist_1$（1，5，9，12），表示这 4 个文档编号的文档含有“走进”这个词汇。同理假定包含“搜索引擎”的文档列表为 $doclist_2$（5，7，9，11），这样同时包含“走进”和“搜索引擎”这两个关键词的文档为 $doclist_1 \cap doclist_2$ =（5，9）。“∩”表示文档列表间的求交运算符。

让我们从具体到抽象，这种在布尔模型中多个查询词而导致的多文档列表求交运算形式化的描述为由一次查询 $Query < T_1, T_2, ..., T_n >$ 引起的查询。首先在倒排索引表检索出文档列表组 $Doclists < doclist_1, doclist_2, ..., doclist_n >$，其中从 T_i 检索出的文档列表为 $doclist_i$。由这次查询返回的结果为 $doclist_1 \cap doclist_2 \cap \dots \cap doclist_n$，查询返回的结果文档中包含了全部查询词。

这里介绍一种多文档列表求交的方法，这种方法称为“最佳归并树算法”。基本思想是越短的文档列表越最早开始参与文档列表的求交过程，越长的文档列表越要推迟计算求交的过程。每次对现存的两个最短的文档列表进行归并，直到归并全部的文档列表，每次归并后减少一个文档列表。因此如果是 n 个文档列表，则需要归并 n 趟。下面通过一个例子来说明这个归并过程，最后得到的一个最佳归并树能够让我们完整地理解这个方法。

假定由多个查询词检索出如下的文档列表组，文档列表按照长度进行升序排序。

doclist1（2，5，7，9，10，15，18）

doclist2（1，2，3，7，8，9，10，12，15，18，22）

doclist3（1，5，7，8，9，10，15，16，18，19，22，23）

doclist4（4，5，6，7，8，9，10，11，15，16，17，18，19，20）

第 1 步归并最短的两个文档列表，即 doclist1 和 doclist2。由于文档列表从倒排索引表中取出，因此文档列表中包含的文档自然地按照文档编号升序排序。这样归并两个有序的文档列表非常

简单且高效，计算方法如下。

（1） 为 doclist1 和 doclist2 分别设置一个当前下标 point_1 及 point_2 并初始化为 0，转（2）。

（2） 如果 point_1 > len（doclist1） 或者 point_2 > len（doclist1），则求交结束；否则转（3）。

（3） 如果 doclist1[point_1]==doclist2 [point_2]，转（4）；否则转（5）。

（4）如果 doclist1[point_1]为一个交集元素，将其存放在返回结果中。point_1++，point_2++，并转（2）。

（5） 如果 doclist1[point_1]> doclist2 [point_2]，point_2++；否则 point_1++，并转（2）。

读者不妨拿出纸笔用这个计算方法一步一步地将求 doclist1 和 doclist2 的交集过程画出来，实际的文档列表求交集实例第 1 步如图 6-4 所示。

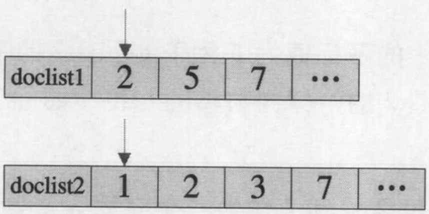


图 6-4　文档列表求交集实例第 1 步

读书笔记

首先初始化下标同时指向两个 doclist 的首个元素，通过比较得到 doclist2[0]大于 doclist1[0]，将指向小值的 doclist2 的下标向右移动一位，如图 6-5 所示。

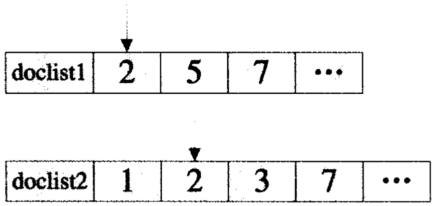


图 6-5 文档列表求交集实例第 2 步

当前 doclist1 和 doclist2 的下标指向的值相等，均为 2。因此 2 作为一个交集元素输出，并且将 doclist1 和 doclist2 的下标同时向右移动一位，如图 6-6 所示。

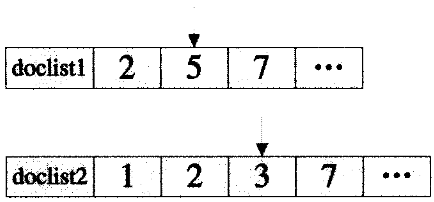


图 6-6 文档列表求交集实例第 3 步

此时 doclist1 的下标值为 5 大于 doclist2 的下标值 3，因此将指向小值的 doclist2 的下标向右移动一位，如图 6-7 所示。

接下来的计算将找到 7 作为一个交集元素输出，完成这个过程可以得到 doclist1 和 doclist2 的交集为（2，7，9，10，15）。

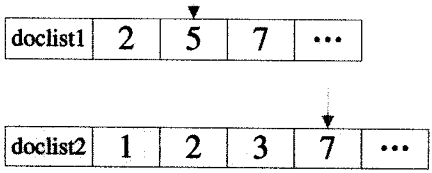


图 6-7　文档列表求交集实例第 4 步

这种计算过程是线性的。因为每次无论何种情况，至少有一个文档列表的指针（point_1 或 point_2）都向右移动一位。只要有一个文档列表指针移动到文档列表尾部，整个计算结束。

此外，很容易证明计算结果的交集长度必然比 doclist1 和 doclist2 都要短，至多和最短的文档列表长度相同（当 doclist2 包含 doclist_1，或者说 doclist1 是 doclist2 的一个子集）。由于文档列表组是按照文档列表长度升序排列的，doclist1 和 doclist2 的交集（不妨称为“doclist1_2”）下一次必定和 doclist3 求交，因为此时文档列表组中最短的两个文档列表必然是 doclist1_2 和 doclist3。

不难得出，最佳归并的顺序如图 6-8 所示。

最佳的归并顺序恰好就是按照文档列表长度排序的顺序，doclist1 和 doclist2 归并。归并结果和 doclist3 归并，归并结果再和 doclist4 归并。

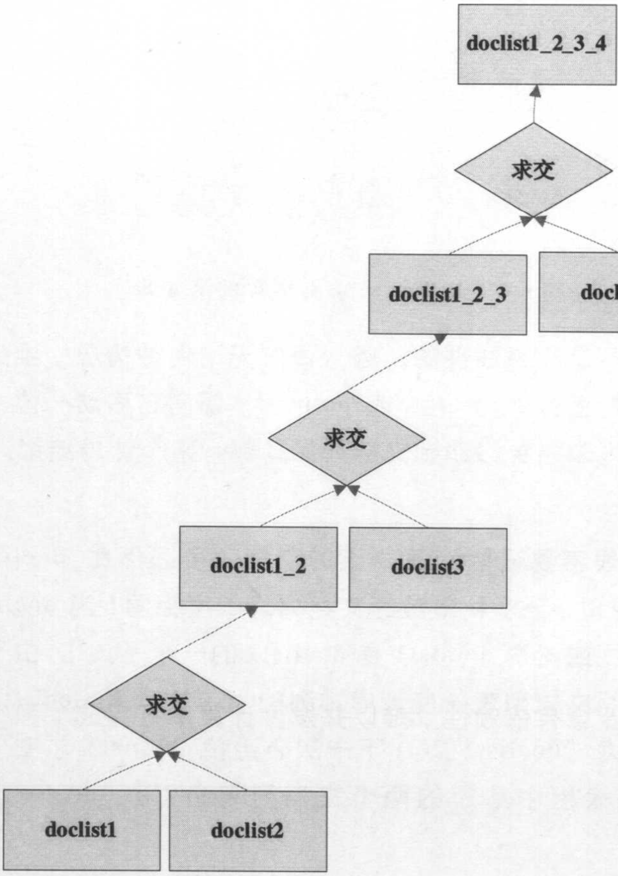


图 6-8 最佳归并顺序

这里值得一提的是，对于那些布尔逻辑表达式中带有“求非”的计算中，“求非”计算优先进行，并且和最短的求与运算的文档列表求非。举个例子，假定出现这样的查询词“搜索引擎 全文检索-检索”，那么这个查询翻译成自然语言为查询包含“搜索引擎”且包含“全文检索”，但不包含“检索”的文档（搜索引擎一般把减号作为去掉某种关键词的查询）。用“搜索引擎”关键词检索出的文档列表为 doclist1，用“全文检索”关键词检索出的文档列表为 doclist2，用“检索”关键词检索出的文档列表为 doclist3。

假定 doclist1 长度大于 doclist2，则无论 doclist3 的长度如何，首先将 doclist2 里去掉和 doclist3 相同的部分，即 doclist2-（doclist2∩doclist3）。这里的减号表示从 doclist2 中去掉 doclist2∩doclist3 后剩余的部分。然后再与 doclist1 求交，这样完整的表达式为 doclist1∩（doclist2-（doclist2∩doclist3））。

这种求文档列表交集的方法的一个最大优点是如果在求最短的两个文档列表的交集时发现为空，即可终止整个求交过程，因为可以断定文档列表组的交集为空。例如，在如图 6-8 的的计算中，如果 doclist1 和 doclist2 的交集为空，则无需继续计算，这 4 个文档列表的求交集的结果必然为空。

当然也存在如下主要缺点。

（1） 计算有依赖性，难以并发。计算串行完成，强行并发会导致大量空间浪费。

（2） 需要在本地开辟额外的空间保存临时的求交结果，总的额外空间大小为第 1 次求得的交集长度。

最后，采用这种归并方法不可避免地存在这样的问题。即在最后一次归并时，必然是一个最短的文档列表和最长的文档列表求交集的过程。如前所述，最后一次归并时，doclist1_2_3 是 3 个文档列表的交集。必然小于或等于最短的文档列表 doclist1，而 doclist4 是最大的文档列表，因此可能出现一个极小长度的文档列表和一个极大长度的文档列表求交。特别是查询词中包含一个低频词（例如“全文检索”）、一个高频词（例如“中国”）和多个中频词时，最后的结果不可避免地出现这种极小文档列表和极

大文档列表求交的情况，从而给计算带来极大的麻烦。

关于文档列表求交的计算方法还有很多，各有优点和缺点。每种方法也包含了很多优化手段，这里不再展开。接下来将进入检索计算的最后一个计算环节——检索结果排序。

检索结果排序

由文档列表组求交集得到的每个文档都需要和查询词一同经过向量化的过程，通过计算文档向量和查询向量的夹角余弦求得向量相似度（一个可以量化的数值），排序就按照这个数的大小关系进行排列。由于搜索结果是海量的，用户也几乎不会耐着性子看完全部的检索结果。有调查表明，大部分的用户使用搜索引擎查询时，在得到搜索结果页后不会向下翻页，而只关注搜索结果的第 1 页。即只实际上需要返回前 n 项结果即可，学术上称此为“top-n 查询”。

由于文档列表按照 PageRank 排序（参阅前面的相关章节），这一点在索引系统中提到过（实际上，文档列表既按照 PageRank 排序，又按照文档编号排序），因此只需要 PageRank 排名靠前的一部分网页拿出来进行这种向量化，然后和查询词相似度的比较即可。而不需要把关键词 doclist 的全部文档都执行这样的计算，这样可以大大降低向量化和向量相似度计算的规模。

例如某个查询词通过布尔模型的求交过程得到 20 万个包含查询词的文档，这里假定只需要查询排名在 256（top-256）以前的结果，因此可以从 20 万个文档中取出一定比例的文档。例如取

出前 5 000 个文档，注意由于这 20 万个文档是按照 PageRank 排序的，所以前 5 000 个文档可以理解为在这 20 万个文档中重要性最高的文档。接下来继续在 5 000 个文档中通过向量化及相似度计算，分别得到这 5 000 个文档和查询词的匹配程度。我们不妨在这里称为"匹配排名"(MatchRank)，表示和查询词的匹配程度。例如使用堆排序或者快速排序这样的经典排序算法对 5 000 个文档的 MatchRank 进行排序，最终取排名前 256 位的文档即可。这样检索出的文档既具有重要性高(PageRank 高)的特点，也具有向量空间模型所要求的相关性强的特点。

在众多排序算法中，由于堆排序具有元素移动少、空间复杂度低并支持 top-n 查询等优点，因此被用来进行检索结果的排序。

堆排序

1991 年计算机先驱奖获得者、斯坦福大学计算机科学系教授罗伯特 弗洛伊德(Robert W. Floyd)和威廉姆斯(J. Williams)在 1964 年共同发明了著名的堆排序算法 HEAPSORT[Willioms 1964]。

首先给出堆排序定义，即大小为 n 的数组 A，下标依次为 1，2，…，n，称为"堆"，当且仅当该数组满足如下性质(简称为"堆性质")：

(1) $A[i] \le A[2i]$ 且 $A[i] \le A[2i+1]$ 或 (2) $A[i] \ge A[2i]$ 且 $Ai \ge A[2i+1]$

若将此数组看做是一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树，即树中任意一个非叶结点的关键字均不大于（或不小于）其左右孩子（若存在）结点的关键字。例如序列（10，15，56，25，30，70）和（70，56，30，25，15，10）分别满足堆性质（1）和（2），故它们均是堆。其中根结点（也称为“堆顶”）的关键字是堆中所有结点关键字中最小者的堆，称为“小根堆”（符合堆性质（1）），如图 6-9 所示；根结点（也称为“堆顶”）的关键字是堆中所有结点关键字中最大者，称为”大根堆”，它符合堆性质（2），如图 6-10 所示。

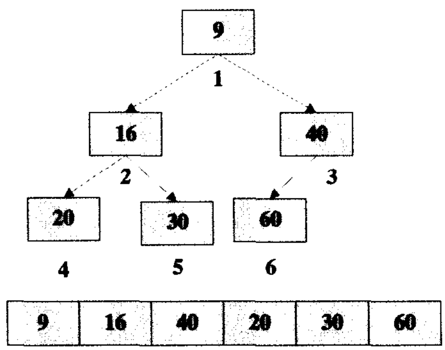


图 6-9　小根堆

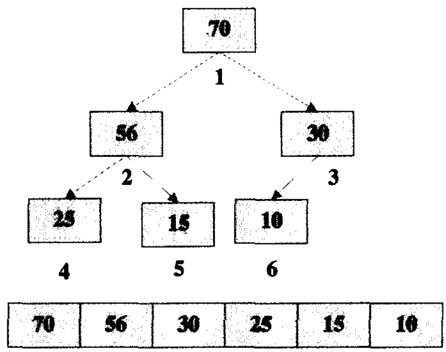


图 6-10　大根堆

接下来以大根堆（可以理解为符合大根堆性质的数组）为例，考察堆排序的过程。

（1）当前堆仅剩下堆顶（堆大小为 1）时堆排序结束，否则转（2）。

（2）堆顶和堆尾置换。

（3）将除堆尾以外的余下元素调整，并符合大根堆性质，转（1）。

不妨考察图 6-10 所示的大根堆的排序过程，堆顶和堆尾置换如图 6-11 所示。

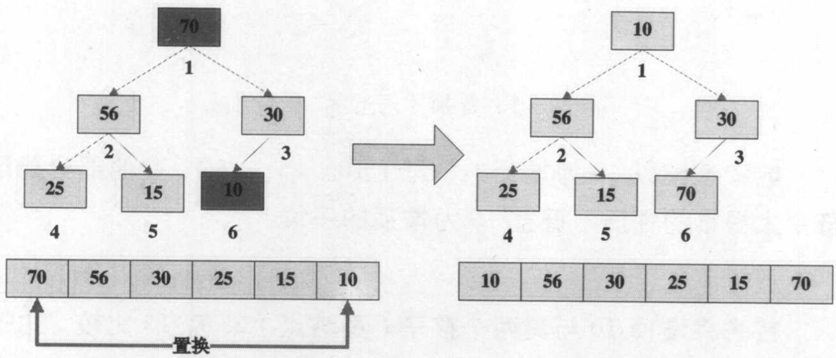


图 6-11　堆顶和堆尾置换

首先将堆顶和堆尾置换（1 号位与 6 号位置换），这样 1 号位存放的元素为 10，6 号位存放的元素为 70，如图 6-11 所示。

读书笔记

接下来需要调整当前堆（注意，这里当前堆不包含 6 号位，即除 70 以外），以符合大根堆的性质。

为调整大根堆，首先将堆顶 10 与其两个孩子 56 及 30 比较。由于 56 是最大值，因此将 10 与 56 置换。即置换 1 号位与 2 号位，如图 6-12 所示。

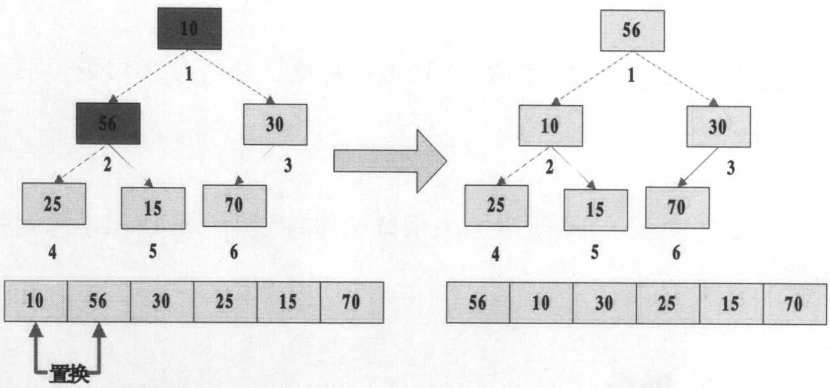


图 6-12　置换 1 号位和 2 号位

继续调整以 10 为堆顶的子堆（10，25，15），其他部分都已符合大根堆的性质，调整 10 为堆顶的子堆。

首先将堆顶 10 与其两个孩子（页结点）25 及 15 比较，由于 25 是最大值，因此将 10 与 25 置换。即将 2 号位和 4 号位进行置换，置换结果如图 6-13 所示。

注意第 1 趟的排序结果将最大值 70 排定顺序，第 1 趟排序后，当前堆为（56，25，30，10，15），依然符合大根堆的如下性质。

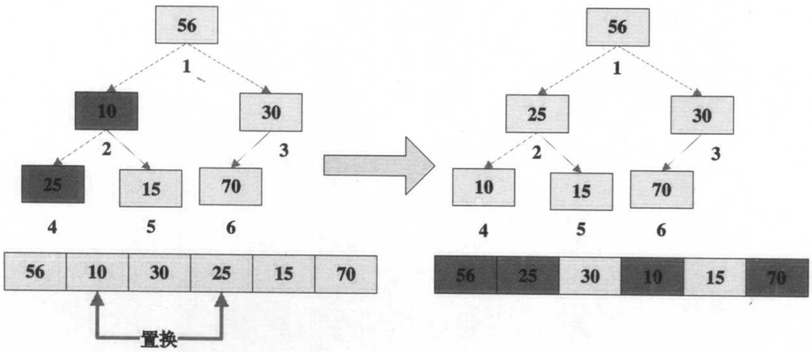


图 6-13　置换 2 号位和 4 号位

1 号位元素大于 2 号位和 3 号位元素（56≥25；56≥30）。

2 号位元素大于 4 号位和 5 号位元素（25≥10；25≥15）。

在第 1 趟排序中，发生移动过的部分如图 6-13 右图中深色块所表示，即 56，25，10，70。而 30 和 15 所在的块并没有发生变化，可见发生移动的数组元素是很少的。可以证明发生移动的元素个数为 0（lgn）数量级的，其中 n 表示当前堆的数目，lgn 为当前堆的高度。每次堆顶和堆尾的置换，或者调整成堆的置换过程至少需要 3 次内存拷贝（两个元素相互交换至少需要 3 次内存拷贝）。因此置换次数较少，节约了大量的内存拷贝。如果是这些元素存放在较大的文件中，则因为节省了 I/O 次数，使得在性能上带来的提升是十分显著的。综上所述，堆排序具有内存拷贝少的特点。

继续将当前堆的堆顶和当前堆的堆尾 5 号位置换（原堆尾为 6 号位，注意每次执行一个操作，当前堆尾位置减 1），调整成大根堆。周而复始，直到当前堆中只剩下堆顶时排序结束，如图 6-14 所示。

读书笔记

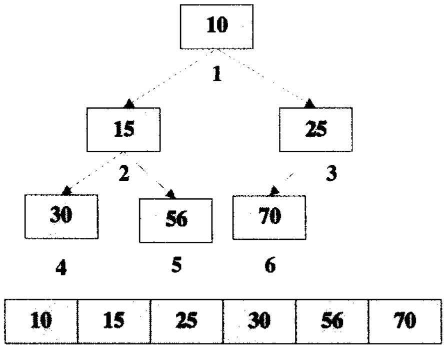


图 6-14　堆排序后的结果

除了具有内存拷贝少的特点，堆排序还具有“就地”排序（in-place）的特点。即排序过程只需要一个置换时的临时元素的存储开销，不需要额外的存储开销。因此堆排序的空间复杂度是 0（1），这个特点可以归纳为空间占用少。

此外，堆排序特别有利于进行 top-*n* 的查询。即仅需要排序前 *n* 个元素，而不需要完全排序所有的元素。例如在如图 6-10 所示的例子中，如果只需要排序前两个元素，那么排序的方法可以改为：

（1） 定义“已经排序的元素个数”变量 sorted_num++，并初始化为 0。

（2） 当前堆仅剩下堆顶（堆大小为 1 时），堆排序结束；否则转（2）。

（3） 堆顶和堆尾置换，sorted_num++。

（4） 若 sorted_num==2，堆排序结束；否则转（5）。

（5） 将除堆尾以外的余下元素调整，并调整当前堆使之符合大根堆性质，转（1）。

排序结束后，数组末位存放最大值，倒数第 2 位存放次大值（second-large），如图 6-15 所示。

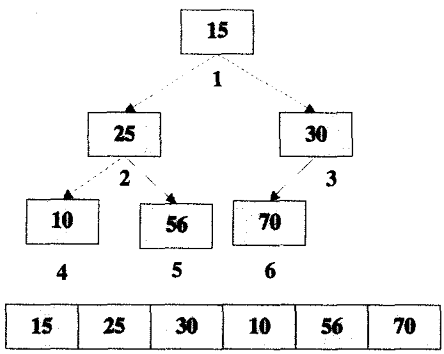


图 6-15　Top-2 排序后的结果

由于其他位置是否有序并不关心，所以排序可以在排好倒数第 2 位后停止。可见堆排序在仅排序前 n 个元素具有计算量低，内存拷贝少的性质。

综上所述，堆排序具有内存拷贝少、额外空间占用少并适合 top-*n* 的排序的特点，因此被用于查询词和文档相关度排序的计算中。

06 第三节 中文自动摘要

中文自动摘要是搜索引擎返回给用户查询结果的重要组成部分，使得用户只需要浏览摘要即可了解需要查询的信息。

自动摘要的发展历史

自动摘要技术比较复杂，涉及自然语言处理（NLP）的核心部分。在详细介绍这部分知识之前，首先来回顾自动摘要这项技术的发展历史。

1958 年，IBM 公司的 H.P.Luhn 首次发表第 1 篇有关自动生成文摘的文章[Luhn 1958]，宣告了该项技术的诞生，至今自动文本摘要的研究已走过了 40 多年的历史。

进入 20 世纪 90 年代，随着电子出版系统和互联网的蓬勃发展，自动文本摘要的价值充分显露出来，越来越受到国内外研究者的重视。

1993 年 12 月，在德国 Wadern 召开了历史上第 1 次以自动文本摘要（Summarizing Text for Intelligent Communication）为主题的国际研讨会。

1995 年，国际期刊 Information Processing & Management 出版了一期题为"Summarizing Text"的专刊，标志着自动文本摘要时代的到来。

1995 年以后，特别是以 Google 为代表的搜索引擎的兴起，自动摘要技术被应用到了搜索引擎的查询结果展示上。

以下我们所说的自动摘要特指搜索引擎领域内的自动摘要提取技术。

自动摘要的含义和实现

自动文本摘要简称"自动摘要"，是从文档中自动提取出的一个正文片断。用户仅仅需要浏览自动摘要就能够了解文档中与查询词相关的部分，进而判断是否值得详细阅读整篇文档。对于同样的一篇文档，其自动摘要对于不同的查询词是不同的。因此自动摘要的计算是实时的，并且是和查询相关的。既需要考虑"效率"，也需要考虑"效果"。

自动摘要在搜索引擎中的实际应用如图 6-16 所示。

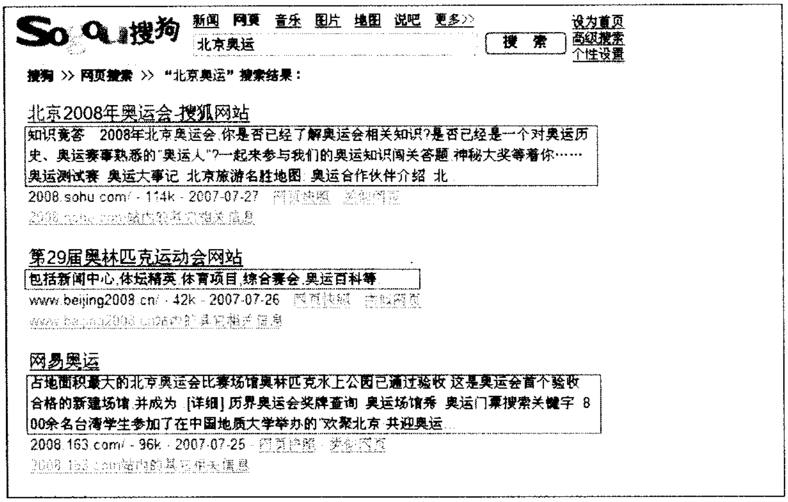


图 6-16　摘要样式

其中方框标出的部分为自动摘要部分，可以看出自动摘要的内容均为实际网页中正文的一个片断。查询词用红色字体标出，这种标识位置信息的技巧在搜索引擎行业称为“标红”。

摘要是搜索结果重要的一个环节，从严格定义上说，它必须包含如下 4 层含义。

（1）**摘要指示性**：摘要必须出现查询词，必须能够指出查询词在文档中的位置。

（2）**摘要描述性**：如果是多个查询词，摘要有限的篇幅最好能够包含全部查询词。如果不能包含全部查询词，也需要尽可能包含权重更高的查询词。

（3）**摘要简洁性**：摘要长度必须控制在一定范围内，既不

能太短，也不能太长。

（4） **摘要完整性**：摘要的句子必须完整，而且摘要的每个组成部分都必须从句子的首部开始，不允许中间断句。

结合第四章“搜索引擎分析系统”中提到的投票算法，以及这里介绍的滑动窗口方法可以较好地解决自动摘要的提取问题，并且满足上面提到的摘要的 4 个特性——指示性、描述性、简洁性和完整性。

滑动窗口实现自动摘要包括如下步骤。

（1） 在文档正文中标记查询词出现的位置（这部分工作事实上在创建倒排索引时已经完成，每个关键词在文档中出现的位置均被标识）。

（2） 从第 1 个查询词开始，取出窗口长度的正文片断作为第 1 个候选窗口。接下来，每次窗口滑动到下一个出现的查询词开始。同样取出窗口长度的正文片断作为候选窗口，直到取完全部的候选窗口。

（3） 在每个候选窗口包含的正文片断中，累计候选窗口中出现的全部查询词的权重作为候选窗口的评分，最终评分高的候选窗口选做自动摘要提取的结果输出。

滑动窗口的方法与第四章中介绍的 Shingle 算法很类似，不过这里每次滑动的距离不定。如果大段的章节没有出现查询词，

则能够一次跳过。由于降低了候选评分的计算量，因此提高了自动摘要计算的效率。

下面通过一个完整的例子来说明整个摘要提取的过程，假设有如下这样一个文档（用斜体表示）。

搜索引擎包含了各个学科的概念和知识，这些学科包含了计算科学、数学、心理学等。特别是数学几乎在搜索引擎的各个系统都大量使用，例如布尔代数、概率论、数理统计等，这些数学知识的应用为搜索引擎解决了一个个的难题，最终使得搜索技术走向成熟。

假设每个滑动窗口取 40 个汉字，标点符号也作为一个汉字。查询词为“搜索引擎 数学”，其权重采用 TF/IDF 方法量化后分别为 6 和 4，那么摘要提取的步骤分为下面几个阶段。

（1）计算查询词在正文中的位置信息，并由<位置，查询词长度，查询词权重>这样的三元组来表示，如图 6-17 所示。

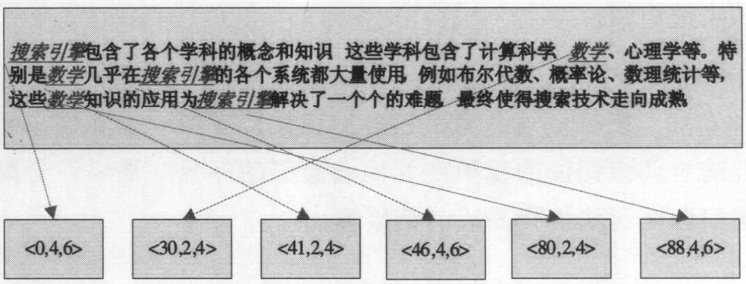


图 6-17　标记查询词在文档中的位置

在标记位置的 3 个分量中，第 1 个分量表示查询词在文档中的位置，第 2 个分量表示查询词的长度，第 3 个分量表示查询词的权重。例如第 1 个位置信息<0,4,6>表示的含义为在文档的位置 0 处出现了一个权重为 6 且长度为 4 个汉字的查询词。

（2） 从文档正文第 1 个查询词出现的位置开始，取窗口长度大小的片断作为第 1 个候选窗口。下面每次窗口滑动到下一个查询词出现的位置，同样取窗口长度大小的片断作为下一个候选窗口。这样循环往复，直到取完所有的候选窗口为止。如果在上一步中标识了 n 个查询词出现的位置，按照这种计算方法，理论上就应该有 n 个候选窗口。参考图 6-17 的查询词位置标识（斜体下划线的词为查询词），可能的候选窗口的起始位置为 0、30、41、46、80 及 88。每个窗口最大取 40 个汉字，最后得到 6 个候选窗口如图 6-18 所示。

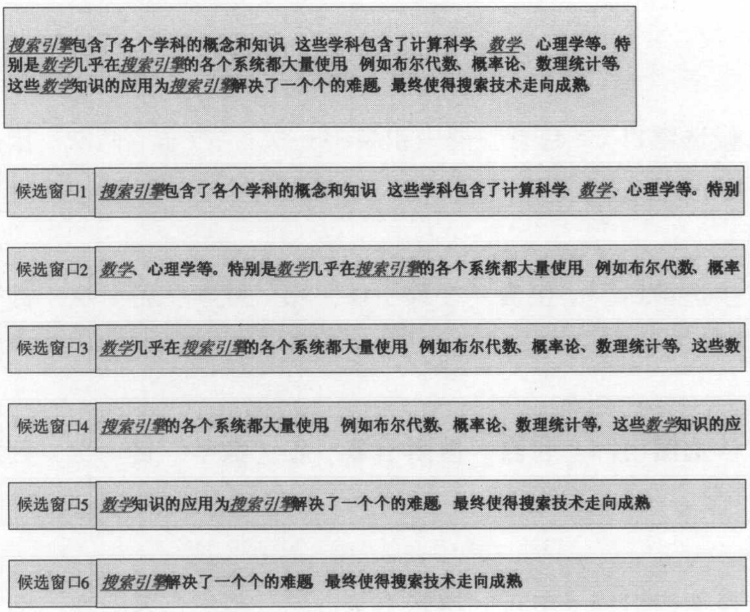


图 6-18　通过滑动得到的 6 个候选窗口

在图中，由于正文包含了 6 个关键词，因此从每个关键词开始都需要取一个窗口，或者理解为窗口每次滑动到下一个关键词开始处。这样共有 6 个候选窗口，每个窗口最大取 40 个汉字。

事实上，还需要设置最小窗口长度，用来保证摘要不会过短。如果最小窗口长度设置为 30，则窗口 6 就不会成为候选窗口，从而避免不必要的计算。

（3） 利用投票算法对各个候选窗口打分，分数最高者为最佳摘要。

如下分别计算每个窗口的得分情况。

候选窗口 1：包含“搜索引擎”和“数学”各一次，因此得到 10 分。

候选窗口 2：包含“搜索引擎”一次，“数学”两次，因此得到 14 分。

候选窗口 3：包含“搜索引擎”和“数学”各一次，因此得到 10 分。

候选窗口 4：包含“搜索引擎”和“数学”各一次，因此得到 10 分。

候选窗口 5：包含“搜索引擎”和“数学”各一次，因此得到 10 分。

候选窗口 6：包含“搜索引擎”一次，因此得到 6 分。

综上，最佳摘要为候选窗口 2 包含的正文片断。回顾摘要包含的 4 个含义，考察候选窗口 2。虽然其中包含了查询词，也具备了一定的长度，而且通过投票算法打分胜出。然而候选窗口 2 表达的句子不够完整，至少没有从一个完整句子的句首开始。

在前面的例子中，假定窗口大小固定不可变，因此取出不完整句子是不可避免的，因此必须从窗口大小入手解决这个问题。即允许窗口大小在一定范围内变动，这种变动主要包含如下两种情况。

（1） 窗口缺失句首部分，尽可能向前多包含几个汉字。例如候选窗口 2，缺少了句首“特别是”这个词，因此窗口可以增大头部包含这 3 个汉字。

（2） 窗口包含了下个句子的开头部分，而并没有实际的意义。例如候选窗口 3 包含了下一个句子的开头部分“这些数”，这些部分可以去掉，因此窗口可以缩小尾部去掉这 3 个汉字。

综上，实际的候选窗口能够通过标点符号进行小范围内的调整，以尽可能地包含一个完整的句子。微调后的候选窗口如图 6-19 所示。

搜索引擎包含了各个学科的概念和知识，这些学科包含了计算科学、数学、心理学等。特别是数学几乎在搜索引擎的各个系统都大量使用，例如布尔代数、概率论、数理统计等，这些数学知识的应用为搜索引擎解决了一个个的难题，最终使得搜索技术走向成熟。

候选窗口1	搜索引擎包含了各个学科的概念和知识，这些学科包含了计算科学、数学、心理学等。
候选窗口2	这些学科包含了计算科学、数学、心理学等。特别是数学几乎在搜索引擎的各个系统都大量使用
候选窗口3	特别是数学几乎在搜索引擎的各个系统都大量使用，例如布尔代数、概率论、数理统计等
候选窗口4	搜索引擎的各个系统都大量使用，例如布尔代数、概率论、数理统计等，
候选窗口5	这些数学知识的应用为搜索引擎解决了一个个的难题，最终使得搜索技术走向成熟。
候选窗口6	这些数学知识的应用为搜索引擎解决了一个个的难题，最终使得搜索技术走向成熟。

图 6-19　微调后的候选窗口

由于使用候选窗口的微调，所以可能会出现如图 6-19 所示的候选窗口重复的情况。例如，候选窗口 5 和候选窗口 6 包含完全一致的正文片断，这些相同的候选窗口需要合并成一个候选窗口。接下来同样采用投票打分的方法，可以得到候选窗口 2 是最佳候选窗口。因此对于“搜索引擎 数学”这样的查询，提取的自动摘要为正文片断“特别是数学几乎在搜索引擎的各个系统都大量使用，例如布尔代数、概率论、数理统计等”。

完成自动摘要的功能后，查询系统就可以拼接检索模块得到的文档和自动摘要模块得到的摘要，从而生成最终的搜索结果页，下面一节中将介绍如何生成搜索结果页。

第四节 生成搜索结果页 06

对搜索引擎的用户来说，搜索结果页是离其最近的部分。搜索结果页的主体包含与查询的相关网页链接（URL）和与查询相关的自动摘要（Automatic Summary），这两个部分的合成还需要一些额外的计算。

生成搜索结果页

第五章提到索引系统中局部倒排文件的分布式部署，如图6-20所示。

这种设计一方面实现了并发的检索，一方面提高了可靠性。正是由于索引结点的这种分布式部署，实际的检索也是在索引结点内部完成。每个索引结点增加一个检索模块从而变成了一个检索结点，一次检索请求引起的计算，直到最后的网页结果生成经历了如图6-21的过程。图中两处检索代理实际上是同一个系统，只是为了方便分开表示。

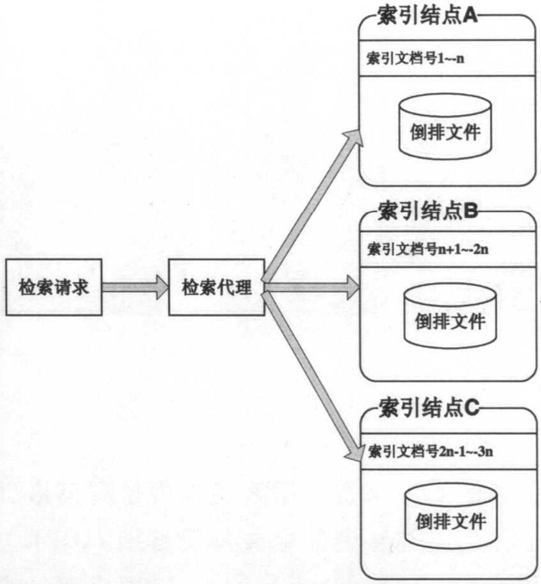


图 6-20 分布式部署

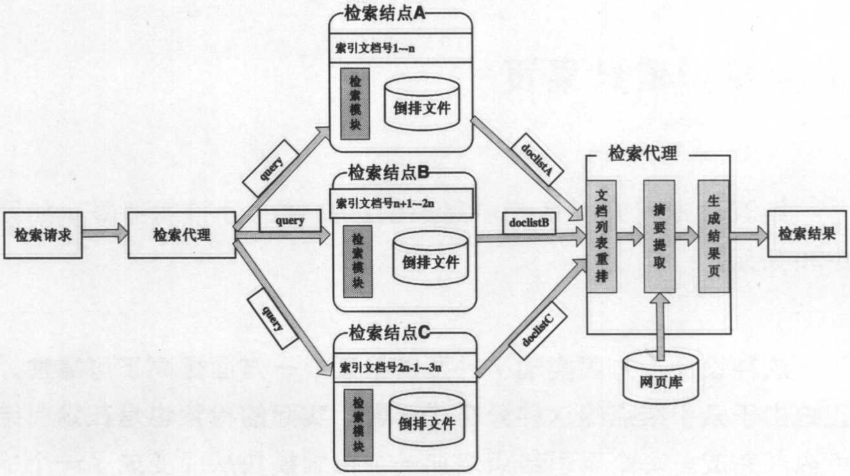


图 6-21 生成搜索结果页的全过程

在图 6-21 所示的搜索结果页生成全过程中主要经历了以下几个步骤。

（1） 检索请求发给检索代理，检索代理进行查询词分词。

（2） 查询词分词后的结果（图中用 query 表示）同时发往各个检索结点。注意这里的检索结点因为具有内部检索功能，因此和图 6-20 中的索引结点不完全相同。通过检索模块的计算，各个检索结点将各自本地倒排文件中检索出的文档列表发给检索代理（不同的索引结点返回不同的文档列表，分别用 doclistA、doclistB 和 doclistC 表示）。

（3） 检索代理重新排序来自各个检索结点的文档，取出排名靠前的 n 个结果文档作为结果页拼接的候选文档。

（4） 通过自动摘要模块从网页库中取出这 n 个文档的摘要信息。

（5） 将（3）和（4）的结果合并，动态生成搜索结果页。

计算出一个搜索结果页需要历经如此复杂的步骤和操作，一个在线的搜索引擎每秒都需要响应相当多的检索请求。如果每次检索请求都经历这样的步骤，显然是不够经济的。与操作系统的缓存设计一样，搜索引擎也为搜索结果页设计缓存。用来减少重复计算，提高效率，下一节将讨论关于搜索引擎结果页缓存的一些设计。

06 第五节 搜索结果页的缓存

在查询系统中，搜索结果页的缓存（Cache）是对搜索"效率"贡献最大的设计。由于缓存中的搜索结果页都是前人查询的结果，因此用户的查询请求如果在缓存中命中（和前人的查询相同），则查询系统直接把缓存中存放的搜索结果页返回给用户。

搜索结果页的缓存

用户在使用搜索引擎进行检索时，查询词可能千差万别。但是如果从大量用户的查询统计上看，总会有一些词汇经常被查询，有些词汇却很少被查询。文献[王建勇, et al.,2001]提出了如下一些结论。

（1） 前 20%的查询词的查询次数约占了总查询次数的 80%。

（2） 查询具有稳定性，查过的词很可能在不久的将来还会被查询。

搜索结果缓存的实现方法和操作系统中提到的 LRU 算法基本一致，下面一起回顾一下 LRU 缓存置换算法。

回顾第三章中提到的网页库设计，对搜索结果页的缓存库必须能够支持随机访问，这一点很重要。如何支持这种随机访问其内部原理和数据库设计很相似，这里不再展开，有兴趣的读者可以参考 B+树等这类能够支持随机访问的索引方式。

有了搜索结果页缓存的设计，搜索引擎查询层就能够大大降低重复的计算量，提高同时响应用户检索请求的能力。具有搜索结果页缓存功能支持的查询系统如图 6-22 所示。

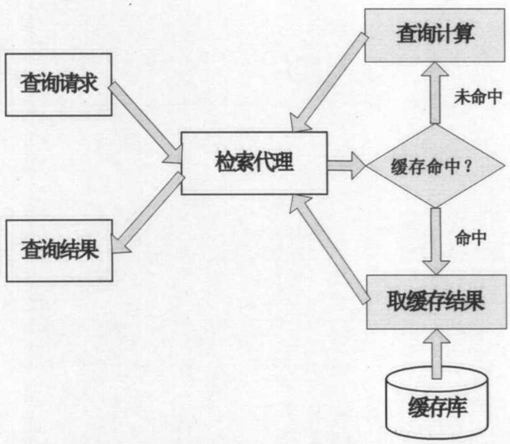


图 6-22　具有搜索结果页缓存功能支持的查询系统

增加了缓存功能后，查询系统可以较少执行实际的查询计算，而采用重用缓存中保存的历史相同的查询结果网页的方法来大大提高查询效率。目前的技术能够达到在缓存中命中 99%的查询，因此用户实际的查询绝大多数情况都是取自缓存的搜索结果页，

读书笔记

这就是搜索引擎为什么能够如此快速地返回查询结果的一个重要原因。

也许是由于搜索结果页缓存的出色设计，在“效率”和“效果”之间的竞争上，“效率”占据了优势。因此近年来，查询系统的研究方向主要在“效果”上，而效果的追求还需要推测用户的查询意图。如果能正确地推测出用户的查询意图，那么对效果的改善可以说是大大有利的。下一节中我们将了解一些关于搜索引擎推测用户查询意图的思想和方法。

第六节 推测用户查询意图 06

推测用户查询意图首先需要对用户的查询做出基本分类，然后通过日志分析及挖掘的技巧对排名进行干预，最终通过大量用户的行为引导最终的排名结果。

查询分类

在实际应用中，用户检索的需求多种多样。根据 Broder 等人对 Alta Vista 搜索引擎的用户日志分析工作，检索可以根据查找信息目的不同分为 3 类，即导航类查询、信息类查询和事务类查询。

2004 年，Yahoo 公司的 Danny 等人在 Broder 工作的基础上将检索类型进行了细化。但总体分类结构则基本保持不变，这也说明了这种查询分类体制的可靠性。

导航类查询的目标是查找一个用户已知的网页（帮助其找到对应的 URL），例如，“软件学报主页”及“清华大学招生简章”

都属于导航类检索的范畴。导航类检索按照查找目标页面的不同细分为特殊需求页面定位任务和主页定位任务两类，主页定位任务的目标页面是站点/子站点的主页；而特殊需求页面定位任务的目标页面则是主页以外的页面。

信息类检索的目标是查找关于某个查询主题的相关信息，如“加强党的执政能力”就可以算做信息类查询。

事务类检索则是用于查找关于某个内容的网络服务，如购物服务、查询服务及下载服务等，典型的样例如“mp3 下载”等。

在这 3 类查询中，不同的查询有着本质的不同。

导航类查询，例如主页的查询可以充分利用锚文本、关键词的位置信息（标题或正文具有不同的价值）及 PageRank 等信息进行查询，总体查询效果是十分理想的。目前绝大多数的搜索引擎都能对这样的查询做到“首条命中”，比如查询“南京大学”，首条必然是南京大学的主页。

而信息类和事物类查询效果就目前来说，和导航类查询差距很大。例如查询“如何买南京到北京的返程票”及“Z50”（南京到北京的一个火车车次），有时搜索引擎返回的查询首页中，10 个查询结果中只有 2 个~3 个是有效的。因此对于查询效果来说，主要的难点在于解决信息和事物类的查询，目前各大搜索引擎都对这类查询效果表现出了极大的关注。

推测信息类、事物类的查询意图

导航类查询通过去除搜索结果中低质量的网页或者排名靠后的方法来提高查询效果，而信息和事务类查询是将搜索结果中高质量的网页排名靠前的方法来提高查询效果，而主要的难点在于对于信息和事务类查询。例如用户查询“Z50”，可能是一个列车的车次，也可能是一款手机的型号，或者是一款数码相机的型号。那么用户查询的到底是什么呢？在推测用户搜索意图上，查询系统做了如下的工作。

（1）从查询日志中得到用户的这类查询中实际点击的URL，并进行排名反馈。

如果在100个查询“Z50”的用户中，50个选择了查看列车车次的URL，10人选择了查看数码相机Z50的URL，5人选择了查看手机Z50的URL，由此反馈给排名系统。在下次再有类似的查询，将和列车车次有关的URL排第一，依次为与数码相机Z50相关的URL，以及与手机Z50相关的URL。这种利用用户的实际点击来进行排名可以满足主流查询用户的需求。

（2）在用户的查询序列中分析查询意图，并给出搜索提示（Query Suggestion）。

通过对用户查询日志的统计分析，例如10%的用户在查询“Z50”后，继续查询了“火车 Z50”或者“Z50 车次信息”，那么可以推测用户在查询Z50后并不满意给出的结果，而使用了更加精确的查询词。因此在搜索提示（一般出现在搜索页的尾部）

中给出搜索提示，帮助用户选择更好的查询词得到搜索效果，同时将在查询“Z50”后的搜索结果中涉及火车车次信息的结果排名靠前。

综上所述，这类信息和事务类查询大多通过事后分析及日志挖掘的技巧将分析结果反馈给排名系统，使得在接下来的排名更加科学。因为具有了用户的点击反馈和查询反馈，所以一个搜索引擎的用户越多，其查询效果就越好。

第七节 查询系统的当前热点和发展方向 06

搜索效果是搜索引擎的命脉，而改善搜索效果的主要途径为查询系统。因此查询系统是搜索引擎中最为热门的话题，目前已成为各大互联网主流会议的主要议题。

查询系统的当前热点

搜索引擎的查询技术发展到当前的水平，主要的发展方向可以归纳为以下几个方面。

（1）推测用户查询意图，这方面主要的工作包括查询纠错（Query correction）、查询推荐（Query Suggestion）及相关搜索等。

（2）能够在某个细分领域进行查询，例如查询行业信息，或者某个领域信息等，这方面工作主要包括垂直搜索及分类搜索等。

（3） 查询结果的优化，这方面工作包括相似结果的聚类、垃圾网页及病毒网页的甄别等。

（4） 提供个性化服务，保存用户的个性化信息并给出可定制的服务，例如可定制搜索服务等。

查询系统，特别是在理解用户查询词、理解用户本身及理解文档方面还有很大潜力可挖。例如向量空间模型还比较初级，没有达到语义分析的水平。再如查询“微机 行情”，文档中只包含了“电脑 行情”的有关信息。由于无法分析“微机”和“电脑”是否是一个概念，因此传统的向量空间模型无法解决这样的问题。

理解查询词、理解用户和理解文档在研究领域取得了一些突破，但是由于计算复杂等原因而难以产品化，离业界的需求还有一段距离。

参考文献

[1] [Baeza-Yates and Ribeiro-Neto,1999] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*: Addison-Wesley- Longman, 1999

[2] [Luhn 1958]P H Luhn . Autornatic creation of literature abstracts . IBM Joumal , 1958 , 2 (4) : 159 ~ 165.

[3] [S. Brin 1998] S. Brin and L. Page. The anatomy of a large-scale hyper textual Web search engine. Computer Networks and ISDN Systems, 30 (1-7) :107-117 , 1998

[4] [Shannon,1948] C.E.Shannon. A mathematical theory of communication. Bell System Technical Journal,27:398-403 , 1948.

[5] [Willioms 1964]Willioms J W.Algorithm 232 "Heap sort"comm. ACM 1964 (7) :347

[6] [王建勇, et al.,2001] 王建勇，单松巍，雷鸣，谢正茂，李晓明“海量 web 搜索引擎系统中用户行为的分布特征及其启示”中国科学 E 辑，vol. 31，No.4, pp. 372-384，2001

[7] [李晓明 2004] 李晓明，闫宏飞，王继民，“搜索引擎——原理、技术与系统”，科学出版社。

第七章

搜索引擎的其他话题

第一节 搜索引擎问与答 07

读书笔记

本书初步介绍了搜索引擎 4 大系统各自的运作原理，以及相互配合的关系。本节通过回答一些有关搜索引擎的基本问题，以从宏观上更好地理解和认识搜索引擎。

为什么搜索引擎的搜索速度这么快

笔者在 Google 搜索引擎中查询“北京奥运”的相关信息，搜索用时 0.21 秒，搜索结果的展示几乎是一瞬间就完成了。信息离我们是如此之近，近到只有 0.21 秒之遥。为什么这些结果只需要花费 0.21 秒的时间就能来到我们的眼前呢？

我们从这样几个方面来解答这个问题。

1. 合理的搜索引擎架构

搜索引擎分为 4 大系统，即下载系统、分析系统、索引系统

和查询系统。这 4 个系统中下载系统、分析系统，以及索引系统称为搜索引擎的“离线部分”(offline part)，只有查询系统称为“在线部分”(online part)。离线部分承担了数据制作的工作，而查询系统承担了数据服务的工作。数据制作的速度没有那么快速，一个网页依次经历抓取、分析和索引入库的漫长过程，这些过程对于用户都是未知的。用户不需要等待这些操作过程，实际用户查询的都是准备好的数据。一般搜索引擎的索引库都有一个做库周期，这个周期内查询系统使用同一个索引库而并不更新。这部分工作主要是缩短做库周期，使得尽可能新的数据能够被检索到。数据服务的速度相当快，得到用户的查询请求后立即完成查询工作，并且返回用户结果。

2. 能够支持快速检索的倒排索引库

倒排索引库是目前为止最优秀并适合大规模数据检索的数据结构，它能够有效且快速地检索出与查询相关的文档。在查询系统中搜索结果页缓存不命中的情况下，也能够快速地检索。当然，关于倒排索引的很多优化设计方面的技巧本书没有展开探讨，这些需要学习计算机操作系统等深入的知识。有兴趣的读者可以参考 Google 搜索引擎的一些设计技巧[S. Ghemawat et al 2003]，进一步了解索引库的设计。

3. 查询系统的一些优化

查询系统做了大量的优化工作，其中最为显著的优化就是查询结果缓存的技巧。对于某个热门查询词，例如“北京奥运”，由于大量的用户会在不同的时段执行这样的查询，并且同一个用户

也会可能连续地查询这样的热门查询词，因此这些查询结果页都是以静态网页的形式被保存在内存中。一旦这些查询词被再次查询，实际上是返回了放在缓存中由前人已经完成的搜索结果页。

综上，搜索引擎“快”的工作主要是由索引系统和查询系统担当，这两大系统共同满足了搜索系统“查得快”的需求。

为什么搜索引擎能够返回那么多的查询结果

笔者在Google搜索引擎中查询“北京奥运”，搜索结果指示，共计查询到6 370 000个结果。看着搜索结果页有无穷的页码显示，不禁叹服当今信息大爆炸带来的巨量数据规模。为收集全这些信息，搜索引擎做的工作主要从两个方面解答。

1.　下载系统抓取了几乎全部的万维网网页

下载系统可以说是搜索引擎的主要驱动源，是搜索引擎发展最为成熟的系统，目前主流的搜索引擎服务提供商在大规模网页下载上已经做到极致。下载系统基本具备了与万维网的变化保持同步的能力，它能够发现万维网上几乎所有网页的消亡、变更和新增，并在网页库中做相应的更新，实时地与万维网的变化保持同步。

2. 索引系统支持存放大量网页的能力

万维网的网页可以说是存放在数以万计的 Web 服务器上，抓取的难度还没有那么大。但是索引如此多的网页对搜索引擎的索引系统无疑是巨大的挑战，索引系统具有超强的分布存储能力，将网页中具有检索价值的信息（锚文本、标题、正文、URL、文档编号，以及关键词出现位置等）充分压缩，并分布式地存放在多台主机的多个硬盘上。关于分布式的设计本书只是初步提到，例如多个索引结点的设计等。分布式的实现主要是多机合作的协议设计，以及一些不同索引结点的分工设计技巧，有兴趣的读者可以参考 SIGIR 会议[Sigir]的一些相关论文。

综上，对于搜索引擎来说，对于海量的网页数据必须做到“抓得了，存得下”，因此下载系统和索引系统共同满足了搜索引擎“查得全”的需求。

为什么搜索引擎总能返回最想要的结果

搜索查询的本质要求就是能够找到用户需要的信息，如果查询到的一大堆结果还需要用户一一选择，则无疑是把用户丢进了信息的汪洋大海，因此搜索引擎在搜索结果排序方面下足了功夫。

搜索引擎把查询分为 3 种类型，即导航类查询、信息类查询和事务类查询。导航类查询采用了 PageRank 排序的方法，将重要性高的网页排名靠前；信息事务类查询充分利用了用户的点击日志和查询日志信息，事后进行分析处理，将大多数人认可的结果网页排名靠前，或者给出搜索提示。因此一个搜索引擎被越多

的人使用，其搜索效果就越好，这种查询反馈的机制被搜索引擎大量地使用。有时信息事务类查询效果不理想时，用户对结果做出的点击，或者使用其他查询词继续查询都会给搜索引擎带来极大的帮助。

当然搜索引擎的查询结果不会总是让所有人满意，在有限的搜索结果展示窗口中只能做到满足绝大部分或者主流用户的需要。例如某个用户带着查询“南京大学校医院”的查询意图，但是输入了“南京大学”这样的查询词。由于大量用户查询“南京大学”的意图是查询“南京大学图书馆”、“南京大学研究生院”及“南京大学招生办”这样的信息，因此在对“南京大学”这个查询词进行查询的结果首页中，前 10 条中都无法包含“南京大学校医院”的查询结果。对于这种情况，用户应该给出更加精确的查询词，例如“南京大学 校医院”。

对于搜索引擎的系统来说，分析系统提供了 PageRank 的宝贵信息，保证了排名具有较高的公正性；查询系统提供了检索模型带来的相关性强、日志分析并反馈排名或给出搜索提示等功能，因此满足了搜索引擎“查得准”的需求。

搜索引擎如何大规模存储网页的

万维网网页的数量是海量的，对搜索引擎来说，大规模网页存储技术是整个领域的巨大成就。在这项主要的技术瓶颈解决后，搜索引擎才能够完成其他关于效果和效率改进的工作。在前面的章节中我们都陆续地提到过一些技术，这里将它们串联起来一一介绍。

在日常生活中，如果有很多产品需要保存，通常有如下几种方法。

（1） 扩大仓库。

（2） 产品放置得尽可能紧凑。

（3）保证重要产品优先存放，可以适当丢弃一些廉价产品。

搜索引擎也采用了类似的思想。

对于扩大仓库的方法，搜索引擎采用了分布式的存储方式。一台机器挂载多块硬盘，多台这样的机器同时存储。因此实际使用的硬盘数量增加，则大大有利于存储更多的信息。第五章中提到的多索引结点策略，第三章中提到的多爬虫合作下载策略，都是这种方法的集中体现。这种方法可以说是最为容易想到，效果也最明显的方法。然而这种粗放式的方法带来的是成本上的巨大耗费，是不是有一些更加经济的方法呢？

产品放置紧凑，在这里可以理解为采用更有效的压缩方法。信息的冗余总是巨大的，第五章中提到的游程编码，继而使用 Variable Byte Coding 编码方式可以大大压缩文档编号，类似的压缩在搜索引擎各个涉及存储的设计中大量采用。例如对网页的存储其实并不是存储原始网页源代码（HTML 语言描述），而是通过分析系统进行了搜索引擎关心的各个域的抽取，最终以网页对象的方式存储，存储的开销也大大降低了。和扩大仓库的粗放方法相比，这种方法充分挖掘存储潜力，使得以有限的存储资源存放

更多网页的方法更加实际有效。

万维网上存在各种各样的网页，大量的垃圾网页和低质量的网页并不是查询的重点，因此对这些网页进行存储是没有必要的。而如何区分哪些网页是高质量的，哪些网页是低质量的，主要由分析系统来完成。例如，垃圾网页和作弊网页识别等。这些虽然没有在分析系统中展开，但是分析系统为减少低质量网页做出了默默的贡献。因此，海量的万维网网页中只有那些质量较高的正规的网页才会被收录。

什么是 SEO

SEO（search engine optimization）的中文意思是搜索引擎优化，由于近些年搜索引擎优化的发展，以至于诞生了搜索引擎优化的行业，从事该行业的人员也被称为“搜索引擎优化工程师”。SEO 和面向搜索引擎的作弊有本质区别，SEO 采用的是合法的方法，充分利用搜索引擎的规则来提高网站或网页的排名。优化主要有以下两个方向。

1. 内容优化

理论上讲，那些 PageRank 值高和网页包含的查询热词多的网页，更有机会出现在搜索引擎的搜索结果页中。因此在网站首页中适当增加关键词有利于排名，网站存在一些来自外部的高质量链接也有助于提高排名。这部分的优化主要面向分析系统和索引系统。分析系统可以计算出高的 PageRank 值；在索引系统中

将网页与更多的关键词相关联，查询系统可以通过向量空间模型的检索方式获得较高的相关性排名。

2. 链接优化

很多网页不能查询，主要由于搜索引擎下载系统没有抓取，网页的深度低，采用动态网页静态化等一系列的面向搜索爬虫友好的技术都有利于改善下载系统抓取的效果。

什么是元搜索引擎

如果说搜索引擎是各个网站的上游，那么元搜索就是搜索引擎的上游，如图 7-1 所示。

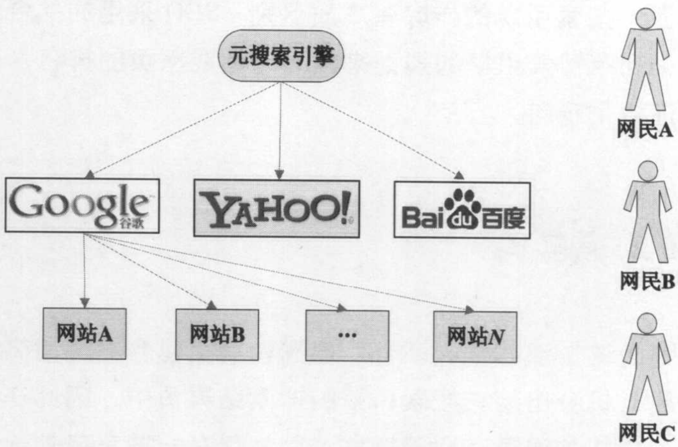


图 7-1　元搜索引擎

搜索引擎索引了各个网页的内容，用户在搜索引擎查询的结果也都来自各个网页，搜索引擎本身并不产生数据。元搜索引擎把用户的查询请求发给主要的搜索引擎，获得搜索引擎的数据然后通过一些特定的结果排序展示给用户。在图 7-1 中，将元搜索引擎的用户看做用户 A，搜索引擎的用户看做用户 B，网站的用户看做用户 C。这些用户都是不同的。网民 C 主要是用浏览行为的网页冲浪方式，或者在网站上执行特定任务；网民 B 主要是搜索行为的用户，带有一定目的性，集中了解一类信息的用户；网民 A 也是搜索行为的用户。然而由于其对搜索引擎的一些排序结果不满意，而特定的元搜索引擎由于在排序结果上照顾了这一类的用户，或者元搜索引擎排序的某种程度上的公正及全面，从而赢得了这一类用户。

虽然元搜索引擎具有实现简单及排序灵活的特点，但是由于其数据完全依赖于搜索引擎的数据，在稳定性和效果的持续改进上都不能做到独立自主，因此至今没有发展出成熟的元搜索引擎公司。

搜索引擎认为的作弊行为是哪些

由于搜索引擎的巨大商业价值，网页或网站排名包含了巨大的商业利益。因此各种面向搜索引擎排名的作弊行为层出不穷，目前对搜索引擎作弊的主要方法分为如下两类。

1. 内容作弊（Content Spamming）

堆砌关键词（Keyword Stuffing）及隐藏文字（Hidden Text）

读书笔记

等这些通过内容作弊而达到搜索排名提高的行为称为“内容作弊”，内容作弊的手段多样。例如在内容中大量充斥着各种查询热词，以及在某些图片后面嵌入一段搜索热词、文字采用白色字体等，这些作弊方法给搜索引擎带来了极大的麻烦。

2. 链接作弊（Link Spamming）

链接作弊主要是针对著名的 PageRank 来进行的，由于指向某个站点的外部链接越多，那么其质量也就越高，PageRank 值也越高，这在分析系统中详细介绍过。这样一些了解 PageRank 排名方法的作弊者在 Internet 上进行大量的链接，这种作弊手段也称为“链接工厂”(Link Farm)。

无论是内容作弊，还是链接作弊都是对搜索引擎健康发展有害的，搜索引擎不得不消耗额外的计算打击和惩罚这种作弊行为。

如何进一步学习和了解搜索引擎发展的最新成果

搜索引擎的全部历史也不过 10 多年，人们不得不惊叹搜索引擎一日千里的发展速度。虽然搜索引擎目前技术上处于稳定和成熟的阶段，但是各个领域的发展还在不断继续。读者如果需要继续了解搜索引擎发展的最新成果，可以通过下面一些途径。

1. 有关搜索引擎的一些世界级顶级会议

- TREC（Text REtrieval Conference），网址为 http://trec.nist.gov/

TREC，中文译为“文本检索会议”，它是文本检索领域人气最旺且最权威的评测会议，由美国国防部和美国国家技术标准局（NIST）联合主办。自从 1991 年举办第 1 届会议起，每年的参与者包括 MIT、Stanford、UCB、微软研究院、Google、IBM 研究院、新加坡国立大学、台湾大学、清华大学、复旦大学、加拿大 Quenes 大学、日本东京大学、香港中文大学及英国城市大学等当今 IT 界一流学府和企业科研机构，并且在不断增加。该会议细分为几大主要方向，即问题回答（QA）、特定领域检索（Legal、Genomics、Enterprise 及 Blog）和传统 Web 检索等。会议负责组织收集并向与会者提供标准的语料库（Corpus）、检索条件和问题集（Query Set），以及评测办法（Evaluation）。与会者则被要求在规定的时间内构造检索系统并提交检索结果（Runs），由会议负责评测各个检索结果的优劣。最终依据评测结果召开大会进行学术交流，发表会议论文。

- SIGIR（Special Interest Group on Information Retrieval），网址为 http://www.acm.org/sigs/sigir/ 。

SIGIR 中文译为“ACM 信息检索专业组”，它是一个国际化的论坛，旨在探讨和交流信息检索领域的最新研究成果、新系统和新技术。会议的主要议题包括信息检索理论与模式、信息检索评估、网络信息检索与数字图书馆，以及文本数据挖掘与信息检索机器学习等。

- SIGMOD（Special Interest Group on Management of Data），网址为 http:// www .sigmod.org/。

SIGMOD 中文译为“ACM 数据管理组”，它是数据库方面最好的会议之一，具有悠久的历史。特别值得一提的是该会议采用了双盲审的制度（double blind review）。在保证公正公平的基础上，提高了会议论文的学术质量。SIGMOD 会议的主要议题几乎涉及了数据库的全部领域，是数据库方面的顶级会议。

- KDD（Knowledge Discovery and Data Mining），网址为 http://www.kdd 2007.com/。

KDD 中文译为“知识发现与数据挖掘国际学术大会”，其前身是 KDD 专题讨论会（Workshop），与会代表均应邀出席。随着 KDD 在学术界和工业界的影响越来越大，KDD 组委会于 1995 年把专题讨论会更名为国际会议，并改为大会代表自愿报名参加。KDD 会议论文主要重点在于数据挖掘方向。

本书的一些参考文献均引自这些会议的论文，可以说这些大会关于搜索引擎的热点很大程度上代表了搜索引擎研究的学术方向。还有一些其他顶级大会，如 WWW 及 VLDB 等的会议论文都极具价值。

2. 参考文献

阅读本书的一些参考文献，这些文献将有助于读者更好地阅读本书，从而达到一个进阶的水平。

第二节 搜索引擎未来的发展 07

当今搜索引擎的各项技术日趋成熟，业界也逐渐将目光从搜索技术本身转移到搜索产品的多样化上，出现了音乐搜索、视频搜索及图书搜索等一系列的搜索产品。相比之下搜索技术仍然在寻求新的突破，本节将从两个方面来介绍搜索引擎未来的发展。

新兴的搜索产品

笔者将当今新兴的搜索产品主要分为 3 类，即多媒体搜索、网页级垂直搜索和对象级垂直搜索。

在这些新兴的搜索产品中，多媒体搜索应该算是其中的“老人”。目前图片搜索、音乐搜索和视频搜索均已被人们所熟知，例如在 Google 图片搜索中搜索“apple”和在 Sogou 音乐搜索中搜索“白桦林”，结果分别如图 7-2 和图 7-3 所示。

视频搜索也逐渐成为业界追逐的焦点，但是由于存在一些特有的技术难度，所以至今还没有较为成熟的视频搜索产品。



图 7-2 图片搜索结果

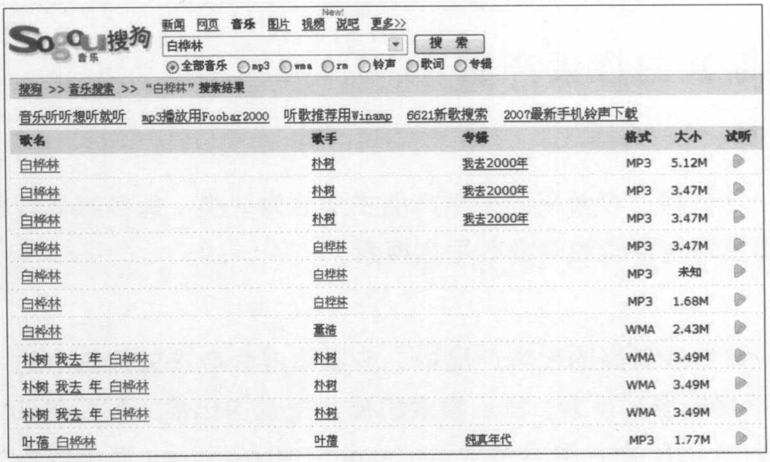


图 7-3 音乐搜索结果

网页级、对象级和垂直搜索究竟是什么关系呢？图 7-4 所示恰当地描述了这种关系，网页级和对象级是针对搜索粒度而言的，

对象级特指那些针对对象属性进行的搜索；而网页级搜索并没有将具体的网页内容对象化。

搜索粒度 / 搜索类型	网页级	对象级
通用搜索	Google	空白
垂直搜索	Google Scholar	Libra

图 7-4 搜索引擎产品的划分

一般认为 Google 的网页搜索属于网页级的通用搜索，这就是本书介绍的传统意义上的搜索引擎，其系统结构和技术要点目前均已达到较为成熟的水平。

Google Scholar（http://scholar.google.com/）属于网页级的垂直搜索，垂直搜索和通用搜索的主要区别在于其限定了某个领域、行业或者话题等。搜索的人群都是固定局限的，例如 Google Scholar 这样的学术搜索面向的是工程技术人员和科研人员。

Libra（http://scholar.google.com/）属于对象级的垂直搜索。目前对象级的搜索是研究的热点，微软公司的 Libra 在这方面做出了积极的尝试，推出了 Libra 这样的论文搜索。由于论文的属性有限，例如作者、摘要及引用等，所以特别适合对象化搜索。

对象级通用搜索目前依然没有成熟产品，这种搜索需要涉及大量信息检索中信息抽取、模式识别及人工智能等高端技术，因而目前仍然没有显著突破，几乎还是一个未开垦的地带。

读书笔记

搜索技术的未来

搜索技术在“效率”和“效果”上已分上下，由于计算机硬件的高速发展，搜索效率已经远远走在了前面。目前搜索技术最让人不满意的还是搜索效果，因此搜索技术的未来依然是效果。笔者认为，改善效果主要有如下 3 个方面。

（1） 理解万维网。

（2） 理解文档。

（3） 理解用户。

万维网瞬息万变，如何保持和万维网变化的同步是爬虫最为头疼的问题，至今依然没有很好的解决方案。在实际使用搜索引擎时还会出现有搜索结果链接“打不开”的情况，在业界称之为“死链”。万维网的信息质量层次不齐，各种垃圾网页及病毒网页层出不穷。在使用搜索引擎时，不断地会打开那些垃圾网页及病毒网页，这都是由于分析系统的分析能力还有待提高，可以说理解万维网是当前也是将来业界的一个主要攻关的技术难点，为此需要引入统计推断及模式识别等技术。

这里说的理解文档，也可以说是理解网页，目前取得了一定的成就。例如 PageRank 就是通过链接关系理解网页的重要性，通过分词及向量空间模型理解文档的意义等。然而在理解文档上还远远不够，至今还不能对这些文档进行无人工参与并以各种粒

度分类网页，甚至分词的水平还有待提高。理解文档才能取得较好的搜索效果，这方面潜力巨大。

理解用户主要从理解用户查询词，从用户的查询历史了解用户的搜索兴趣等。只有一方面理解用户，一方面理解文档，搜索引擎才能知道哪些文档和用户的查询词最匹配。例如，一个用户搜索“清华大学”，搜索引擎发现这个搜索请求来自台湾省（通过IP 地址识别），因此推测其很可能是搜索台湾省的清华大学，因而给出台湾省清华大学的搜索结果。可见抓住一切可能具有的用户信息，充分理解用户是改善搜索效果的重要方法。

信息的载体是文字，由于普通人搜索的内容一般是本国语言的信息。所以作为搜索引擎这样的以信息聚合技术为鲜明特点的软件产品，还需要积极打破这种语言的限制。Google 在这方面做出了一些有意义的尝试，然而机器自动翻译属于世界级难题。不过可以预见，如果将来的某一天搜索引擎能够打破这种语言的限制，那么由于搜索引擎的翻译，我们甚至可以不懂法语也可以了解法国巴黎街头发生的一件好人好事，这将是多么美妙。

综上所述，搜索引擎还需要开发更多的搜索产品，还需要修炼出更加强劲的搜索技术。考虑到搜索引擎仅 10 年的历史就能够取得现在的辉煌成就，我们有理由期待在不远的将来，搜索引擎还会给人们带来更多的惊喜。

参考文献

[1] [S. Ghemawat et al 2003]Sanjay Ghemawat , Howard Gobioff , Shun-Tak Leung, The Google file system, Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22 , 2003 , Bolton Landing , NY , USA.

[2] [Google Img] http://images.google.cn/

[3] [KDD]http://www.kdd2007.com/

[4] [SIGIR]http://www.acm.org/sigs/sigir/

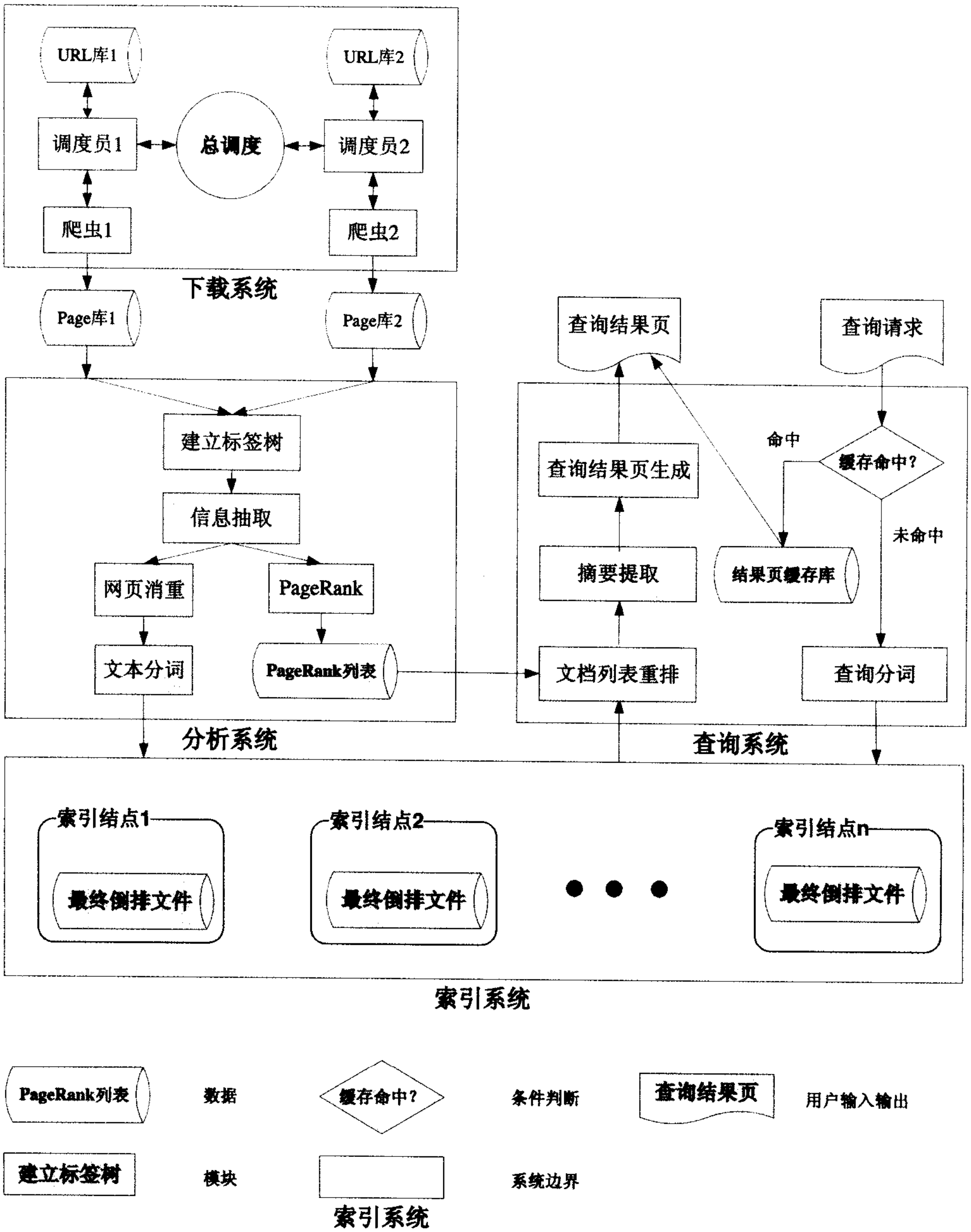
[5] [SIGMOD]http://www.sigmod.org/

[6] [Sogou mp3] http://mp3.sogou.com/

[7] [TREC] http://trec.nist.gov/

附录 A

搜索引擎系统结构全观图


URL库1
URL库2
调度员1
总调度
调度员2
爬虫1
爬虫2
下载系统
Page库1
Page库2
建立标签树
信息抽取
网页消重
PageRank
文本分词
PageRank列表
分析系统
查询结果页
查询请求
查询结果页生成
命中
缓存命中？
未命中
摘要提取
结果页缓存库
文档列表重排
查询分词
查询系统
索引结点1
最终倒排文件
索引结点2
最终倒排文件
索引结点n
最终倒排文件
索引系统
PageRank列表
数据
缓存命中？
条件判断
查询结果页
用户输入输出
建立标签树
模块
系统边界
索引系统